AFAL-TR-73-203
Volume I

AVIONICS PROCESSOR-CONTROLLER CONFIGURATION STUDY

TECHNICAL REPORT-VOLUME I

L. J. Koczela
Electronics Group of Rockwell International
Anaheim, California 92803

TECHNICAL REPORT AFAL-TR-73-203 VOL. I

DISTRIBUTION STATEMENT

Distribution limited to U.S. Government Agencies only; test and evaluation results reported; February 1972. Other requests for this document must be referred to Air Force Avionics Laboratory (AAM), Wright-Patterson Air Force Base, OH 45433.

AIR FORCE AVIONICS LABORATORY
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AVIONICS PROCESSOR-CONTROLLER CONFIGURATION STUDY

TECHNICAL REPORT-VOLUME I

L. J. Koczela

DISTRIBUTION STATEMENT

Distribution limited to U.S. Government Agencies only; test and evaluation results reported; February 1972. Other requests for this document must be referred to Air Force Avionics Laboratory (AAM), Wright-Patterson Air Force Base, OH 45433.

FOREWORD

This Final Engineering Report was prepared by the Electronics Group of Rockwell International, Anaheim, California. The work was accomplished under USAF Project 6090 entitled "Avionics Data Handling Technology", Task 01 entitled "Avionics Information Processing" and contract No. F33615-72-C-1973 entitled "Avionics Processor-Controller Configuration Study." The work was administered under the direction of Mr. J. E. Camp, Air Force Avionics Laboratory, AFAL/AAM, Wright-Patterson AFB, Ohio.

This report covers work conducted from 1 July 1972 to 30 June 1973 and was submitted by the author 30 April 1973.

This technical report has been reviewed and is approved for publication.

COZIER S. KLINE
Colonel, USAF
Chief
System Avionics Division

ABSTRACT

This report presents the results of a study to configure an advanced multiprocessor for an avionics system. An advanced strategic bomber avionics system was selected as representative of an advanced avionics system application and the computational requirements for this system were defined. The prototype laboratory version of an advanced multiprocessor developed by Burroughs Corporation under Air Force Avionics Laboratory sponsorship was examined and applied to the avionics system. It was found that the Burroughs multiprocessor offers a very flexible and adaptable design. Several improvements were noted to improve its performance and several design modifications were noted which are required in order to apply the design to the avionics system. The resultant configuration showed that mechanization of the computer system, using state-of-the-art technology, for an advanced strategic bomber avionics system is feasible with the Burroughs multiprocessor concept. This report is also being published as Autonetics internal report C72-812/201.

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS (Cont)

TABLES

Table | Page

1. INTRODUCTION

This report presents the results of a study to configure an advanced multiprocessor for an avionics system. The Air Force Avionics Laboratory has developed a prototype of an advanced multiprocessor in recent contracts with Burroughs Corporation (Ref 1). The objective of this study was to define the computational requirements of an advanced avionics system, investigate the Burroughs multiprocessor in light of the requirements imposed by an avionics system, and configure a computer system for the avionics application using the Burroughs multiprocessor design.

The study was divided into four principal tasks as shown in Figure 1-1:

Task 1 - Requirements Analysis

Task 2 - Module Definition and Analysis

Task 3 - Configuration Definition

Task 4 - Physical Characteristics

Section 2 describes the results of defining the computational requirements. Section 3 contains a brief summary describing the Burroughs multiprocessor concept. Section 4 presents the results of the analysis of the Burroughs multiprocessor with regards its capabilities and limitations in an avionics system environment. Section 5 presents the configuration for the avionics system using the Burroughs multiprocessor modules. Finally, Section 6 contains an estimate of the physical characteristics of the Burroughs multiprocessor for the central computer of the avionics system.

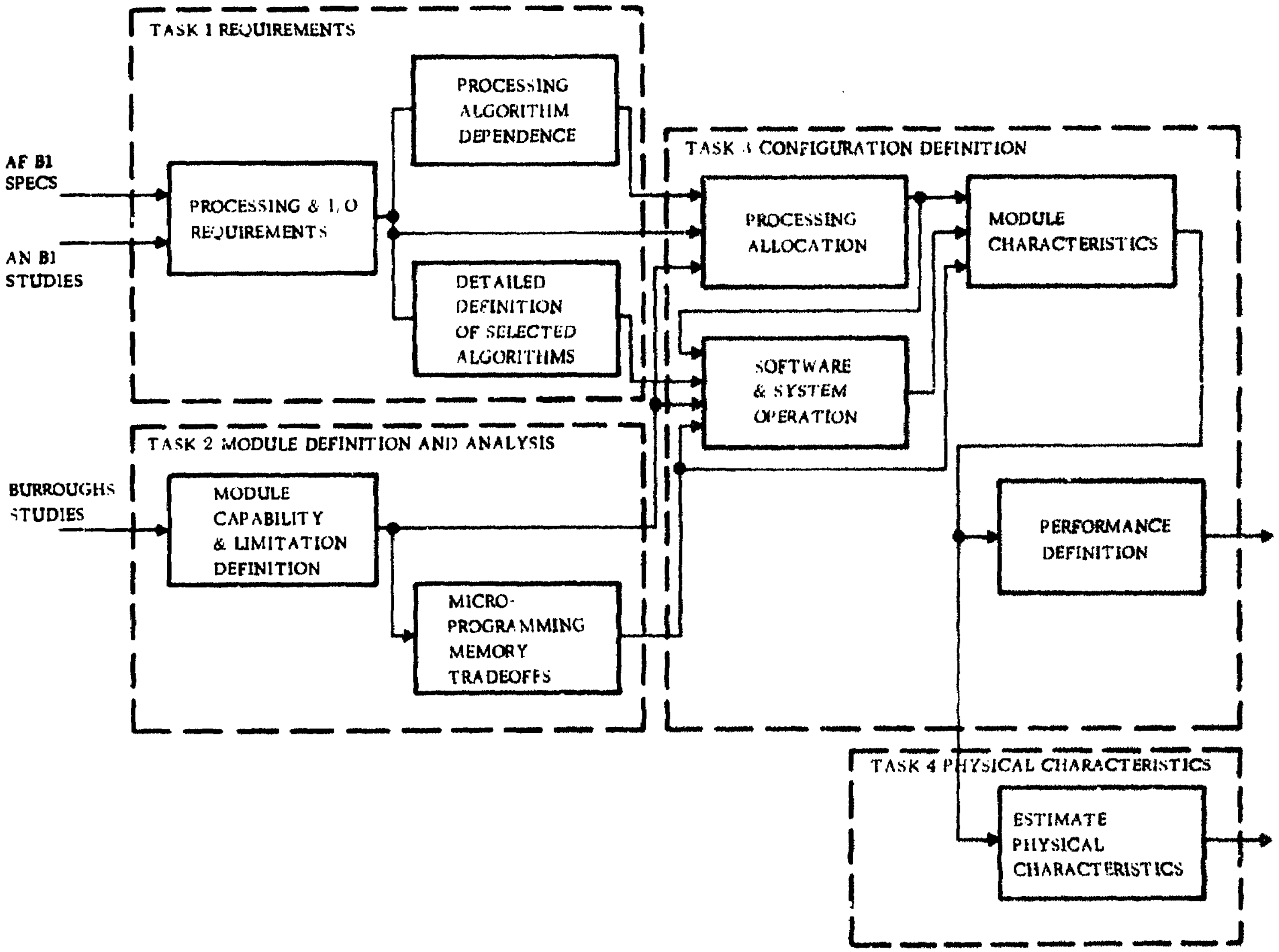

TASK 1 REQUIREMENTS
AF B1 SPECS
AN B1 STUDIES
PROCESSING & I/O REQUIREMENTS
PROCESSING ALGORITHM DEPENDENCE
DETAILED DEFINITION OF SELECTED ALGORITHMS
TASK 2 MODULE DEFINITION AND ANALYSIS
BURROUGHS STUDIES
MODULE CAPABILITY & LIMITATION DEFINITION
MICRO-PROGRAMMING MEMORY TRADEOFFS
TASK 3 CONFIGURATION DEFINITION
PROCESSING ALLOCATION
SOFTWARE & SYSTEM OPERATION
MODULE CHARACTERISTICS
PERFORMANCE DEFINITION
TASK 4 PHYSICAL CHARACTERISTICS
ESTIMATE PHYSICAL CHARACTERISTICS

Figure 1-1. Avionics Processor - Controller Study Flow

2. REQUIREMENTS ANALYSIS

The purpose of the Requirements Analysis was to provide a set of realistic advanced strategic bomber avionics digital data processing requirements. From these requirements and the results of the module analysis task, an appropriate Avionics Processor-Controller subsystem can be configured.

2.1 ANALYSIS APPROACH

2.1.1 Advanced Strategic Bomber (ASB)* Avionics System Definition

The B-1 Avionics System was selected as representative of the advanced strategic bomber avionics system for the purpose of this study. The B-1 avionics configuration was initially established by the September 1971 RFP for the B-1 Avionics System Interface Contractor (ASIC). This baseline configuration has been amended as the result of amendments to the RFP, the new avionics subsystems proposed by Autonetics in response to the RFP, and by changes to the B-1 avionics system since award of the ASIC contract.

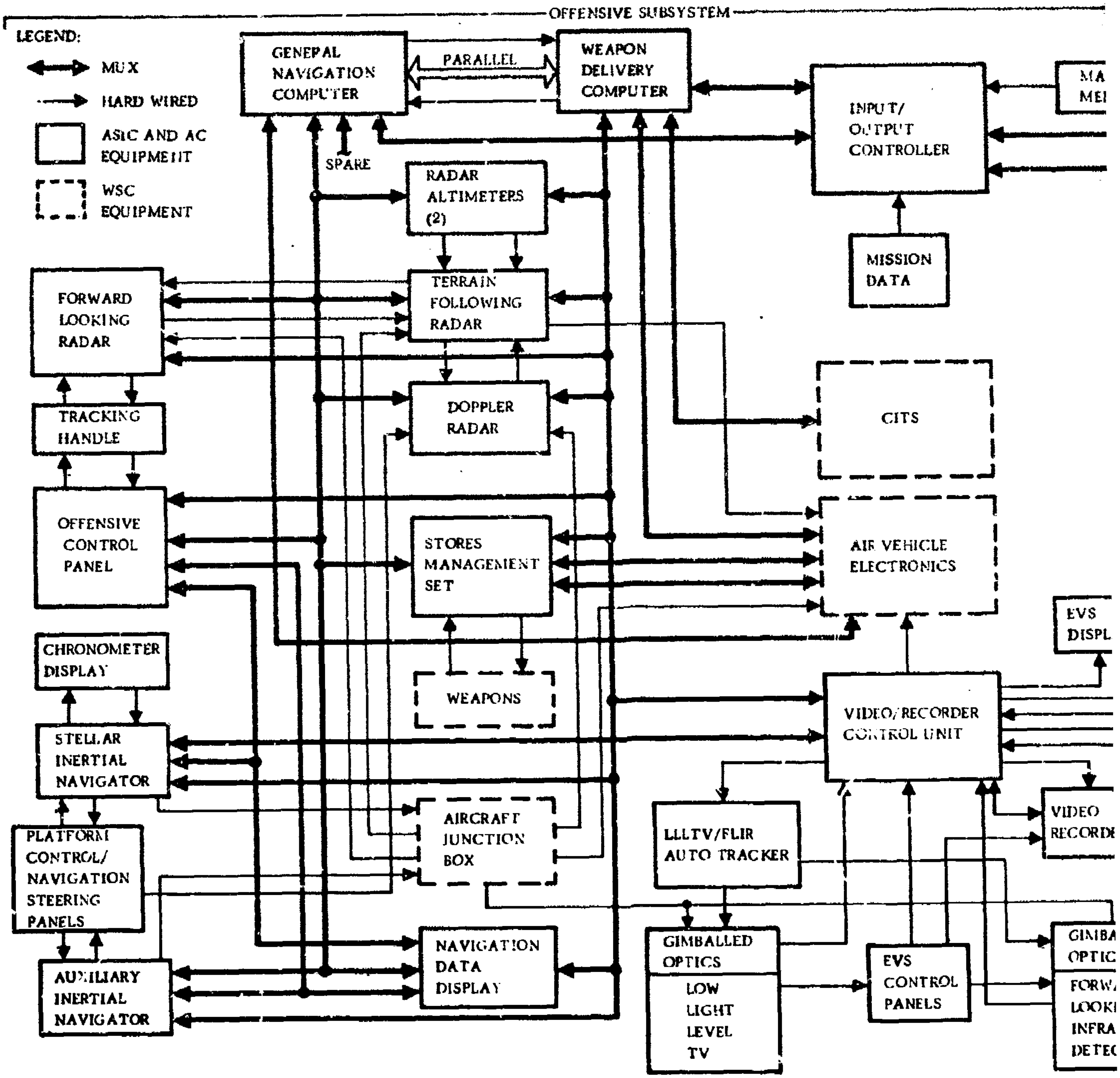
The B-1 avionics configuration was further modified in response to the goal of this study in two definite areas. First, the offensive and defensive subsystems defined by the modified RFP were combined into one avionics system. Secondly, the pre-processors currently in the B-1 avionics hardware were assumed to be not included in the avionics configuration.

The resultant B-1 avionics configuration established for this requirements analysis is oriented about a central digital data processing system. The final avionics configuration for this study will be defined later in this report when pre-processors are allocated as a result of the Configuration Definition task.

Some figures are included for background information. Figure 2-1 is a block diagram of the B-1 Avionics System as proposed by Autonetics in response to the last AF amendments to the RFP. Note that central computers with specific functions are contained in this system. In the offensive subsystem they are the General Navigation Computer (GNC) and Weapon Delivery Computer (WDC). Besides the primary functions contained within each computer, backup mission essential functions (not including CITS) are also contained in each computer. These backup functions are activated in either computer should the other computer fail. The General Defensive Computer (GDC) is the sole central computer in the Defensive Subsystem. Note that no backup processing of GDC functions is available in case of GDC failure. For this reason, Autonetics proposed an alternate configuration as shown in Figure 2-2. Both GDC and Central Integrated Test Subsystem (CITS) backup mission essential functions can then be provided in identical central computer subsystems as well as primary functions within the Defensive Subsystem.

Figures 2-3 and 2-4 show the Avionics Multiplex (AMUX) assignment contained within the Autonetics proposal for the offensive and defensive subsystems, respectively. The hardware subsystems are grouped in response to functional capabilities and data transmission loading considerations.

*ASB shall be used for "Advanced Strategic Bomber" in this report.


OFFENSIVE SUBSYSTEM
LEGEND:
MUX
HARD WIRED
ASIC AND AC EQUIPMENT
WSC EQUIPMENT
GENERAL NAVIGATION COMPUTER
PARALLEL
WEAPON DELIVERY COMPUTER
INPUT/ OUTPUT CONTROLLER
MA MEI
SPARE
RADAR ALTIMETERS (2)
MISSION DATA
FORWARD LOOKING RADAR
TERRAIN FOLLOWING RADAR
TRACKING HANDLE
DOPPLER RADAR
CITS
OFFENSIVE CONTROL PANEL
STORES MANAGEMENT SET
AIR VEHICLE ELECTRONICS
CHRONOMETER DISPLAY
EVS DISPL
WEAPONS
STELLAR INERTIAL NAVIGATOR
VIDEO/RECORDER CONTROL UNIT
PLATFORM CONTROL/ NAVIGATION STEERING PANELS
AIRCRAFT JUNCTION BOX
LLLTV/FLIR AUTO TRACKER
VIDEO RECORDE
AUXILIARY INERTIAL NAVIGATOR
NAVIGATION DATA DISPLAY
GIMBALLED OPTICS
LOW LIGHT LEVEL TV
EVS CONTROL PANELS
GIMBA OPTIC
FORW LOOKI INFRA DETEC

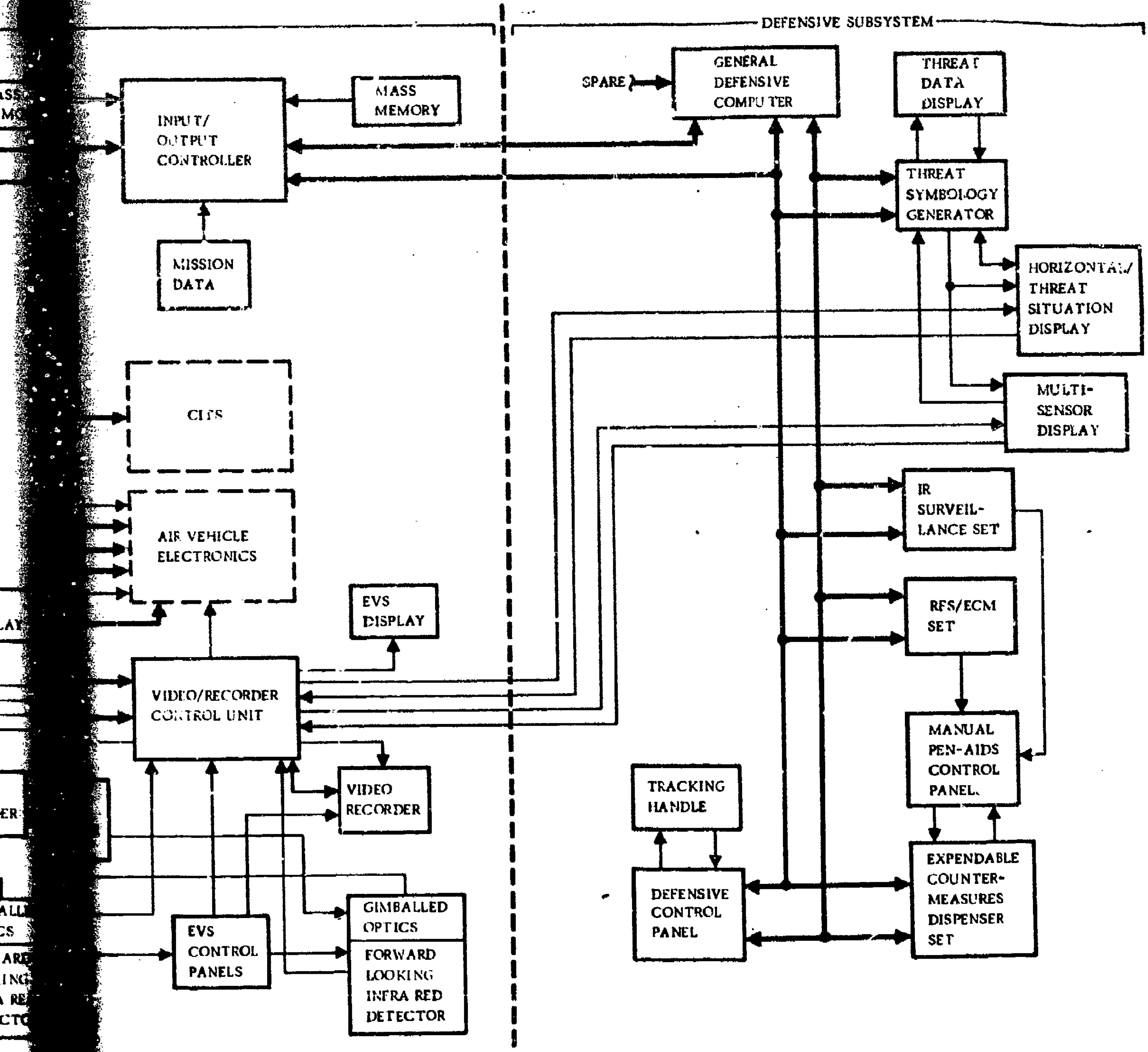


Figure 2-1. B-1 Avionics System Block Diagram

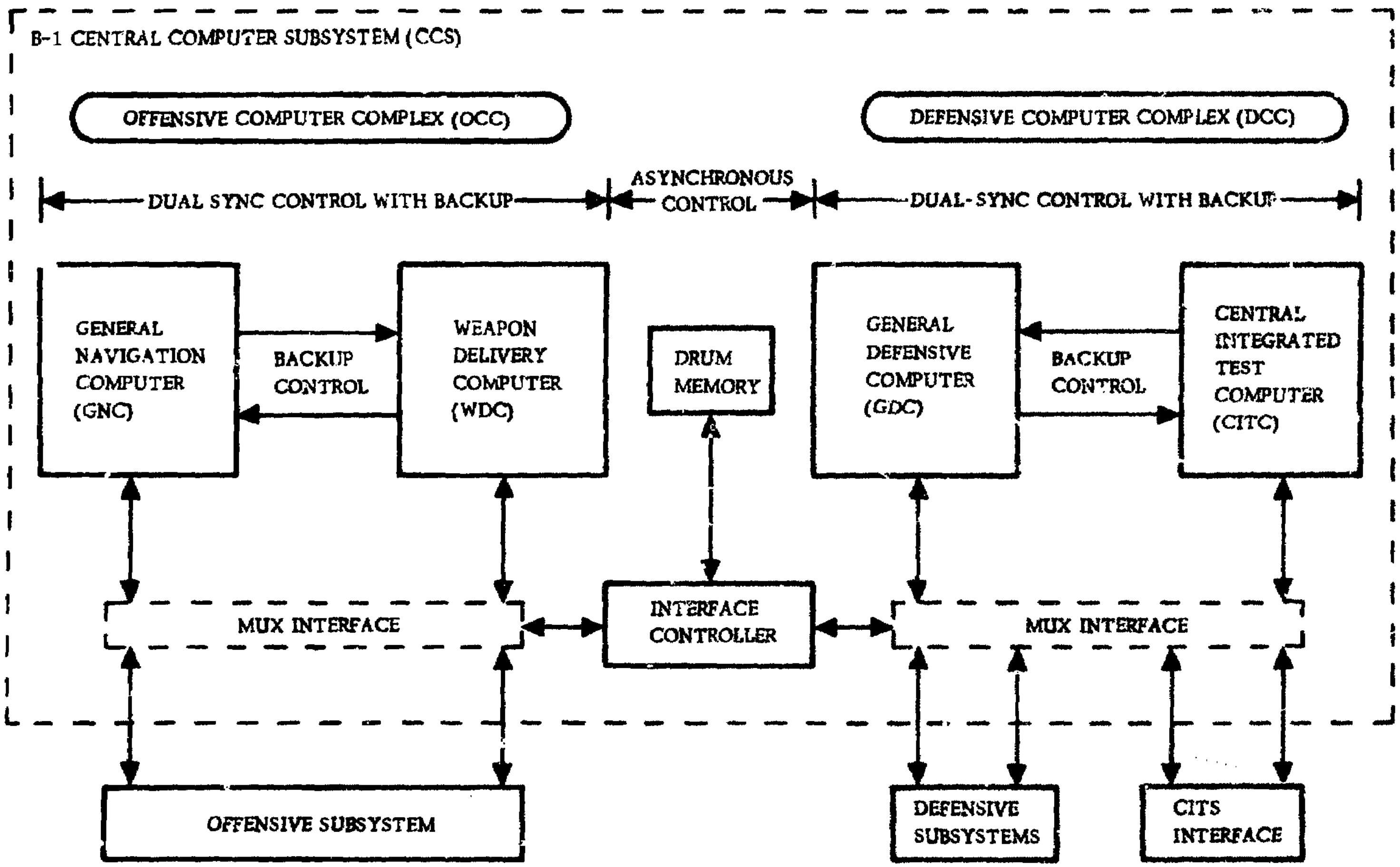

B-1 CENTRAL COMPUTER SUBSYSTEM (CCS)
OFFENSIVE COMPUTER COMPLEX (OCC)
DEFENSIVE COMPUTER COMPLEX (DCC)
DUAL SYNC CONTROL WITH BACKUP
ASYNCHRONOUS CONTROL
DUAL-SYNC CONTROL WITH BACKUP
GENERAL NAVIGATION COMPUTER (GNC)
BACKUP CONTROL
WEAPON DELIVERY COMPUTER (WDC)
DRUM MEMORY
GENERAL DEFENSIVE COMPUTER (GDC)
BACKUP CONTROL
CENTRAL INTEGRATED TEST COMPUTER (CITC)
MUX INTERFACE
INTERFACE CONTROLLER
MUX INTERFACE
OFFENSIVE SUBSYSTEM
DEFENSIVE SUBSYSTEMS
CITS INTERFACE

Figure 2-2. Alternate B-1 Central Computer Subsystem

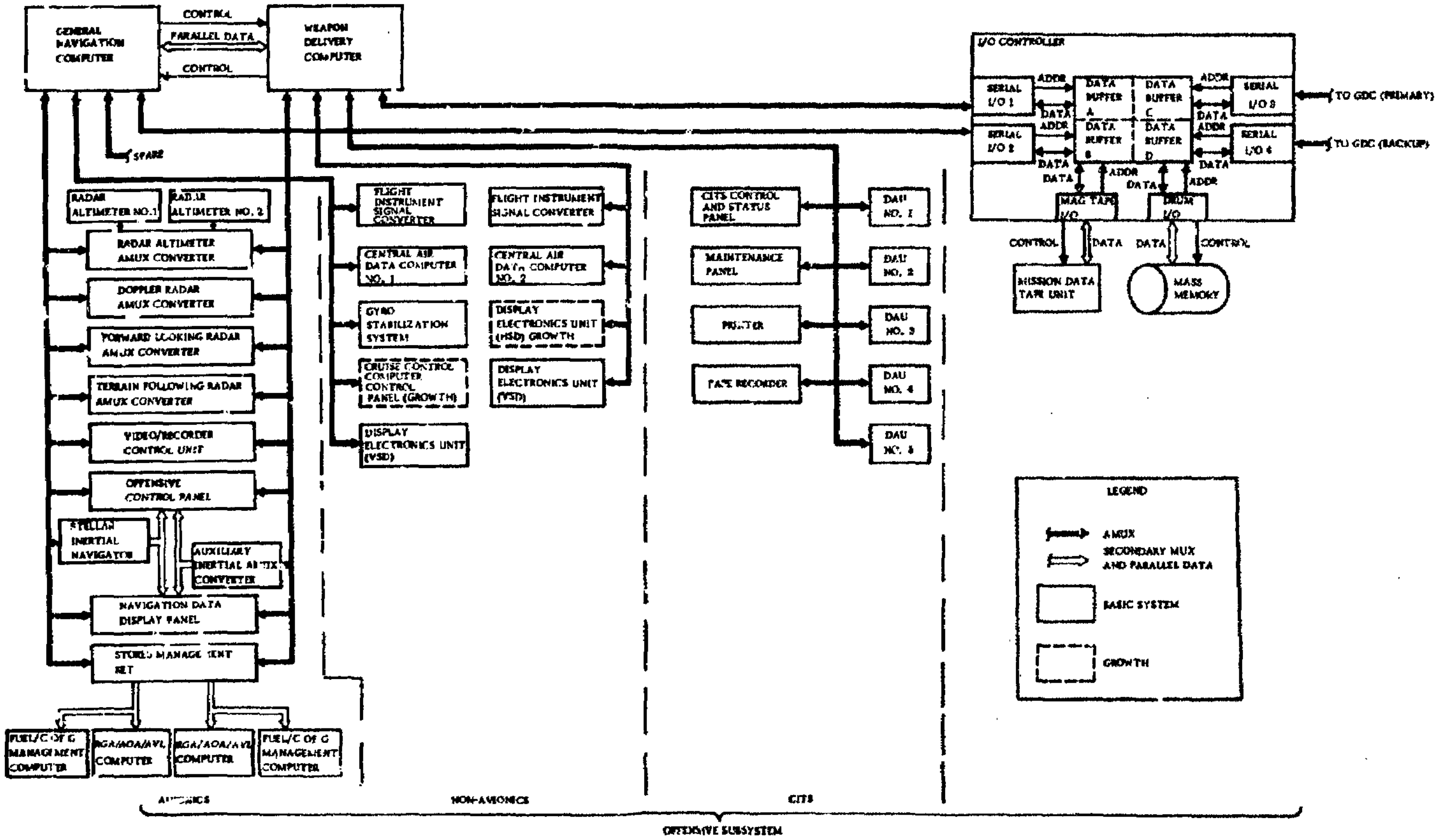

GENERAL NAVIGATION COMPUTER
CONTROL
PARALLEL DATA
CONTROL
WEAPON DELIVERY COMPUTER
SPARE
RADAR ALTIMETER NO. 1
RADAR ALTIMETER NO. 2
RADAR ALTIMETER AMUX CONVERTER
DOPPLER RADAR AMUX CONVERTER
FORWARD LOOKING RADAR AMUX CONVERTER
TERRAIN FOLLOWING RADAR AMUX CONVERTER
VIDEO/RECORDER CONTROL UNIT
OFFENSIVE CONTROL PANEL
STELLAR INERTIAL NAVIGATOR
AUXILIARY INERTIAL AMUX CONVERTER
NAVIGATION DATA DISPLAY PANEL
STORES MANAGEMENT SET
FUEL/C OF G MANAGEMENT COMPUTER
RGA/AOA/AVL COMPUTER
RGA/AOA/AVL COMPUTER
FUEL/C OF G MANAGEMENT COMPUTER
FLIGHT INSTRUMENT SIGNAL CONVERTER
CENTRAL AIR DATA COMPUTER NO. 1
GYRO STABILIZATION SYSTEM
CRUISE CONTROL COMPUTER CONTROL PANEL (GROWTH)
DISPLAY ELECTRONICS UNIT (VSD)
FLIGHT INSTRUMENT SIGNAL CONVERTER
CENTRAL AIR DATA COMPUTER NO. 2
DISPLAY ELECTRONICS UNIT (HSD) GROWTH
DISPLAY ELECTRONICS UNIT (VSD)
CITS CONTROL AND STATUS PANEL
MAINTENANCE PANEL
PRINTER
TAPE RECORDER
DAU NO. 1
DAU NO. 2
DAU NO. 3
DAU NO. 4
DAU NO. 5
I/O CONTROLLER
SERIAL I/O 1
SERIAL I/O 2
ADDR
DATA
DATA BUFFER A
DATA BUFFER B
DATA BUFFER C
DATA BUFFER D
SERIAL I/O 3
SERIAL I/O 4
TO GDC (PRIMARY)
TO GDC (BACKUP)
MAG TAPE I/O
DRUM I/O
CONTROL
DATA
MISSION DATA TAPE UNIT
MASS MEMORY
LEGEND
AMUX
SECONDARY MUX AND PARALLEL DATA
BASIC SYSTEM
GROWTH
AVIONICS
NON-AVIONICS
CITS
OFFENSIVE SUBSYSTEM

Figure 2-3. Offensive Subsystem Computer Interface

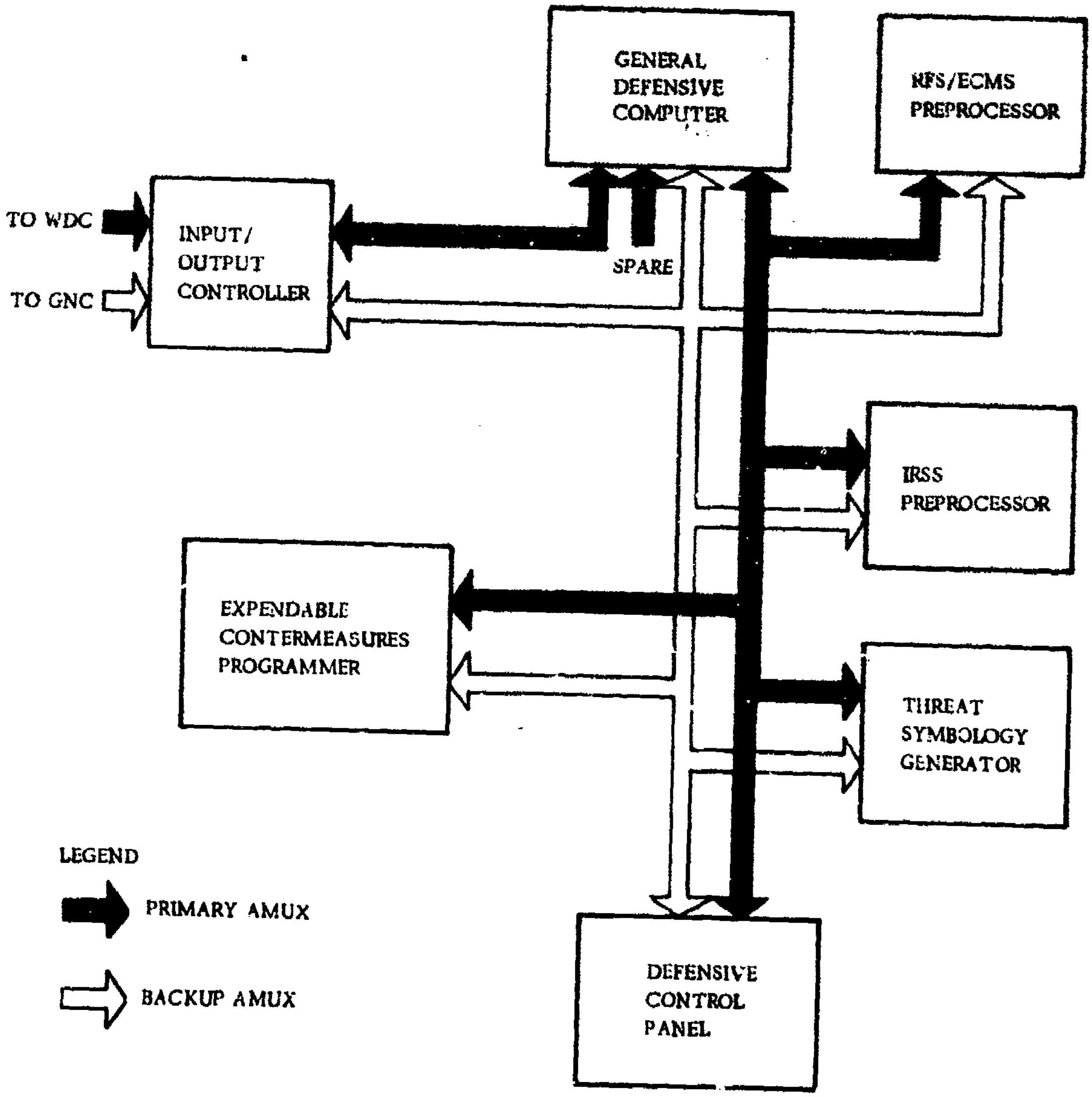


Figure 2-4. Defensive Subsystem AMUX Interface

Finally, Figure 2-5 shows the ASB avionics configuration established for this study. A definition of the subsystem hardware abbreviations is contained in Table 2-1. Note that the AMUX configuration shown is not the actual implementation design required, dual data transmission over separate AMUX lines is required for fail operational AMUX capability. Also, while the primary function interface is shown, e.g., Navigation with IMU 1, other functions may have data communication requirements, e.g., CITS with IMU 1.

In defining the avionics configuration, the data transmission in and out of the crew capsule was minimized for physical considerations. The hardware subsystems included in the cockpit (crew capsule) are identified in Figure 2-5 to allow association with Offensive and Defensive Subsystem operators (rear cockpit) as well as front seat operation.

2.1.2 Processing Requirements Definition Approach

The overall processing tasks were first grouped into major functional elements:

1. Navigation
2. Steering
3. Target/Checkpoint Acquisition
4. Weapon Delivery
5. Penetration Aids
6. Mission Data Management
7. CITS (Central Integrated Test System)
8. Executive

The association of the hardware subsystems with their major interfacing and supporting functions is shown in Figure 2-5. The intent of the functional subsystem grouping is to provide a maximum of subsystem operational capability with a minimum of interfunction data transmission.

The detail processing requirements were estimated for each of the identified functional groups. Each major function was broken down into several processing tasks. The processing tasks were determined based on interfacing hardware processing requirements, the basic selectable modes of operation, and the complexity of the total function.

The definition of a processing task as used in this requirements analysis is a major processing segment executable as one contiguous element at a specific rate. No attempt was made to define extremely small tasks within a specified rate dependent upon multiple conditions and mode selection. Most of the conditional logic is assumed to be contained within a defined processing task. This approach was taken since it is generally the most efficient avionics processing implementation approach. Extreme

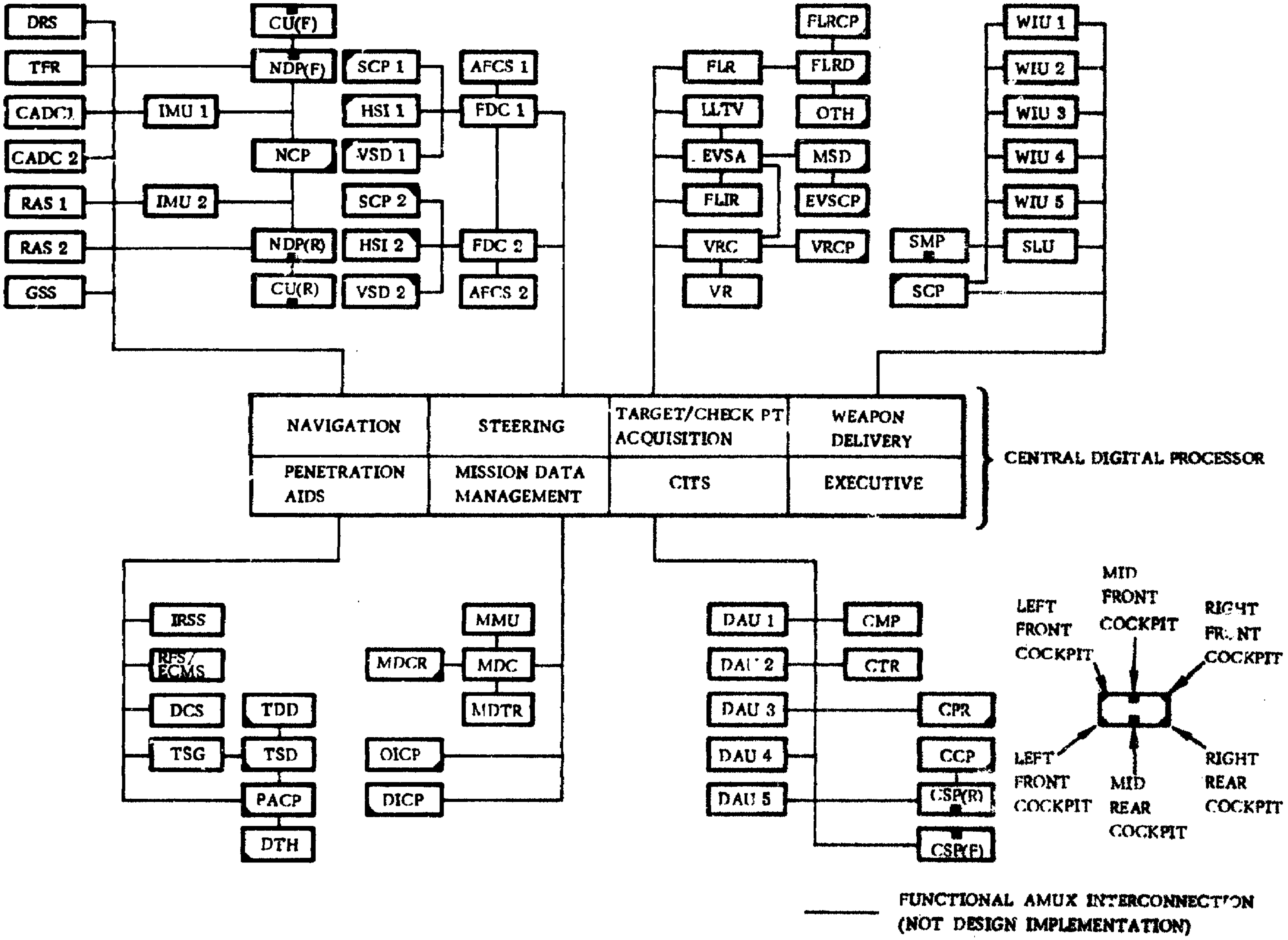


Figure 2-5. Avionics System Function/Hardware Interconnection for the Avionics Processor Controller Study

Table 2-1. ASB Avionics Subsystem Equipment List

Navigation Subsystems

- Inertial Measurement Unit (IMU), 1 and 2
- Doppler Radar Set (DRS)
- Radar Altimeter Set (RAS), 1 and 2
- Navigation Control Panel (NCP)
- Navigation Display Panel (NDP), Front and Rear
- Chronometer Unit (CU), Front and Rear
- Terrain Following Radar (TFR)
- Central Air Data Computer (CADC), 1 and 2
- Gyro Stabilization Subsystem (GSS)

Steering Subsystems

- Flight Director Computer (FDC), 1 and 2
- Automatic Flight Control Subsystem (AFCS), 1 and 2
- Steering Control Panel (SCP), 1 and 2
- Horizontal Situation Display (HSD), 1 and 2
- Vertical Situation Display (VSD), 1 and 2

Target Checkpoint Acquisition Subsystems

- Forward-Looking Radar Control Panel (FLRCP)
- Forward-Looking Radar Display (FLRD)
- Forward-Looking Radar (FLR)
- Low Light Level Television (LLTV)
- Forward-Looking Infrared (FLIR)
- Offensive Tracking Handle (OTH)
- Multisensor Display (MSD)
- EVS Control Panel (EVSCP)
- EVS Autotracker (EVSA)
- Video Recorder (VR)
- Video Recorder Controller (VRC)
- Video Recorder Control Panel (VRCP)

Table 2-1. (Cont)

Weapon Delivery Subsystems	
	Stores Management Panel (SMP)
	Store Logic Unit (SLU)
	Weapon Interface Units (WIUs), 1 through 5
	Stores Consent Panel (SCP)
Penetration Aids Subsystems	
	Radio Frequency Surveillance/Electronic Countermeasure Set (RFS/ECMS)
	Infrared Surveillance Set (IRSS)
	Penetration Aids Control Panel (PACP)
	Threat Symbology Generator (TSG)
	Threat Situation Display (TSD)
	Threat Data Display (TDD)
	Defensive Tracking Handle (DTH)
	Dispensables Control Set (DCS)
Mission Data Management Subsystems	
	Offensive Integrated Control Panel (OICP)
	Defensive Integrated Control Panel (DICP)
	Mission Data Cartridge Reader (MDCR)
	Mission Data Tape Recorder (MDTR)
	Mass Memory Unit (MMU)
	Mission Peripheral Controller (MPC)
Central Integrated Test Subsystems	
	Data Acquisition Unit (DAU), 1 through 5
	CITS Maintenance Panel (CMP)
	CITS Tape Reader (CTR)
	CITS Printer (CPR)
	CITS Control Panel (CCP)
	CITS Status Panel (CSP), Front and Rear

modularization based upon multiple conditions is inefficient in an avionics implementation for several reasons. First, a high overhead in executive task scheduling determination logic and data shuffling is required due to the high number of multiple conditions and executable tasks as well as intertask data transfer. Second, an excessive amount of documentation must be generated and maintained for the separately identifiable tasks and associated input and output data. Also, additional testing and verification must be performed at the detailed task level to validate task computation as well as intertask data transmission. The task level selected allows mode and submode branching to be performed within specified tasks.

The processing requirements were determined by performance of the following tasks:

1. Estimation of the hardware/function interface signals and required data transmission rates.

2. Estimation of the function/function interface signals and required data transmission rates.

3. Segmentation of each major function into processing tasks executable at specific processing rates.

4. Segmentation of each processing task into identifiable subtasks.

5. Estimation of the intra-function signals between processing tasks within each function.

6. Estimation of the number of operations (instructions) and data (combined parameters, variables, and constants) for each task and subtask. The number of operations per iteration was computed based on assuming 80 percent of the total instructions per subtask being executed during a given computational iteration. The total number of operations per second was then computed based on the required iterations per second. The total task processing requirements were then summed by adding subtask totals. If the task was determined to contribute to the worst case throughput requirement, it was tagged for subsequent inclusion in the requirements summation for the major function.

7. Prerequisite processing tasks were identified for each processing task.

8. The total memory and throughput requirements for each major function were determined by totalling the individual task requirements.

9. A block diagram of each major function was generated to give a visual representation of the overall function operation. Major generic subfunctions, consisting of multiple processing tasks were identified. The major interface signals between subfunctions and external hardware and other functions were identified.

10. The off-line mass memory requirements were also estimated.

2.2 ASSUMPTIONS

1. The processing requirements are defined for a fault tolerant computer. Fault tolerant requires that the computer will continue to operate in at least a limited capability mode after failure of one of each type of module in the computer. An on-line backup mechanization is required which will allow at least a minimum continuous computation for critical functions during the computer reconfiguration time. This reconfiguration time exists from detection of a fault to the loading and subsequent initialization of an off-line backup mechanization in the remaining operational computer elements.

   The minimum on-line backup mechanization must consist of critical elements of all critical functions. While a detail estimate of this backup mechanization was not made (with exception of the Steering function), a gross estimate is contained in Table 2-5.

2. The memory and throughput processing requirements were estimated for the individual processing tasks without regard to memory word length or instruction type. The total processing requirements were converted to an absolute number of memory words and operations per second in terms of a conventional computer by applying appropriate weighting factors. A 16-bit word length conventional computer was assumed. The total number of 16-bit memory words and operations per second were computed from weighting factors derived from FB-111A/F-111D digital computer complex experience (based on experience with the IBM 4 PI avionics computer).

3. In estimating the throughput requirements, 80 percent of the actual instructions (excluding subroutines) within a task were considered to be executed per iteration. The total operations per iteration were then determined by adding any common subroutine usages to the 80 percent actual instruction count. Multiplication by the required iteration rate gave the throughput in operations per second per task.

4. The worst case throughput requirements were considered for both subtasks within processing tasks and processing tasks within a major function. Within a processing task, the worst case subtask throughput was calculated for tasks with mutually exclusive subtasks. Among processing tasks, the rationale was used to include only worst case processing task throughput requirements derived from worst case operating modes.

5. The processing requirements for the preprocessors presently planned for inclusion in the B-1 Avionics System are included. These preprocessing requirements are presently included in the inertial navigator units, the non-avionics CITS, the SMS, the IRSS and the RFS/ECMS.

6. No data format conversion is required within the computer other than binary to decimal and decimal to binary. All analog to digital and digital to analog conversion is assumed to be performed by the avionic subsystem hardware.

7. In addition to the basic requirements imposed by the functions described in this report, it is required that the on-line system contain a 50 percent spare capacity. It is also required that the total on-line system be capable of a 100 percent growth factor.

8. The throughput requirements for common subroutines are based on FB-111A/F-111D digital computer complex experience. These requirements in terms of operations per second are given in Table 2-2 and are used in calculating the subtask and task throughput requirements.

Table 2-2. Subroutine Throughput Requirements

Subroutine	Operations
Sine/Cosine (SC):	30 (per sin or cos)
Arctangent (ATAN):	70
Square Root (SR):	40
Matrix Multiply (MX):	70
Euler Transformation (EUL):	80
Binary to Decimal Converstion (BCD):	70
Decimal to Binary Conversion (DEC):	70

9. The weighting factor to determine the on-line memory requirements in 16 bit words is based on a 70 percent short (16 bit) and 30 percent long (32 bit) format mix of both instruction and data words. This mix is derived from FB-111A/F-111D digital computer complex actuals.

Weighting Factor	No. of 16-Bit Words	Total Words (16 Bit)
.70	1	.70
.30	2	.60
		1.30 - weighting factor to determine total memory in 16-bit words

10. The throughput requirements represent a mix of instruction types. The mix is given in Table 2-3 and is based on FB-111A/F-111D digital computer complex actuals. This mix may be used later in the study when the relative execution time of the various instruction types is determined.

Table 2-3. Throughput Weighting Factor

Instruction Type	Execution Frequency Weighting
Load/Store	.38
Add/Sub.	.13
Multiply	.10
Divide	.01
Shift	.08
Branch	.14
Logic/Misc.	.16

2.3 REQUIREMENTS SUMMARY

2.3.1 ASB Avionics System Processing Requirements

The ASB Avionics System central processing and input/output data transmission requirements are summarized in Table 2-4. These requirements include both on-line spare provisions and growth capabilities required in the central computer. As explained above, these requirements assume that all the processing for the major functions is performed in a central computer.

Table 2-4. ASB Avionics System Processing Requirements Summary

	Memory (16 Bit Words)	Throughput (KOPS/Sec)	Input/Output Rate (16 Bit Words/Sec)
Primary Program	111,000	688[1]	37,744
On-Line Spare (50%)	55,500	344	18,872
Subtotal	166,500	1,032	56,616
Growth Capability (100%)	166,500	1,032	56,616
	333,000	2,064	113,232

[1] Represents a mix of various types of operations as given in Table 2-3

2.3.2 Processing Requirements of Central Computer Functions

A summary of the processing requirements for the functions mechanized in the central computer is given in Table 2-5. These requirements are independent of word length (the total requirements given in Table 2-4 were converted into equivalent 16 bit words) and represent a mix of various types of operations as indicated in Table 2-3. It should be noted at this point that the values for the executive and on-line back-up program should be considered only as rough estimates. These requirements are summarized from detailed tabulations of processing requirements for each of the functions and are given in Appendix A. A portion of these detailed tabulations is given in Section 2.4 for illustrative purposes.

Table 2-5. Processing Requirements of Central Computer Functions

	Memory (Words) [3]	Throughput (KOPS/Sec) [4]
1. Navigation	13,217	126
2. Steering	2,900	15.20
3. Target/Checkpoint Acquisition	1,810	57.84
4. Weapon Delivery	10,245	83.24
5. Penetration Aids	25,500	220.36
6. Terrain Following/Avoidance[1]	0	0
7. Mission Data Management	4,285	17.52
8. Mission and Traffic Control [2]	0	0
9. Central Integrated Test Subsystem	18,541	66.56
10. Executive (Estimate)	5,000	50.00
11. On-Line Back-up Program (Estimate)	4,000	50.00
Total	85,498	686.72

[1] Included in the Navigation function

[2] Growth function

[3] Independent of word length, a multiplying factor of 1.3 will convert these into 16 bit words as explained in Section 2.2

[4] Represents a mix of instruction types as indicated in Table 2-3

2.3.3 Input/Output Data Transmission Requirements

A summary of the input/output data transmission requirements is given in Table 2-6. This table gives the requirements by rate groups for both input and output to the central computer. The detailed tabulation of the input/output requirements used to derive this summary table is given in Appendix A.

The input/output information transferred between the central computer and the avionics hardware subsystems must be transmitted over the Avionics Multiplex (AMUX) subsystem. The transmitted data must conform to the format established by the Multiplex Interface Modules (MIMs) which provide the interface between the AMUX and adjoining subsystem hardware. Electrical and physical compatibility must also exist between the subsystem hardware and the interfacing MIMs. A portion of the specification for the MIM is included in Section 4 of this report where the interface to the ASB multiplex system is considered.

2.3.4 Mass Memory Requirements

The mass memory requirements are given in Table 2-7. The requirements are tabulated for five categories of information storage:

Table 2-6. Input/Output Data Transmission Requirements Summary

	Data Transmission Rate (Transmissions/Sec)	Number of 16 Bit Words per Transmission (Words/Transmission)	Number of 16 Bit Words per Second (Words/Sec)
Function Input	64	8	512
	32	193	6,176
	16	948	15,168
	8	12	96
	4	14	56
	2	743	1,486
	1	90	90
Function Output	64	8	512
	32	40	1,280
	16	569	9,104
	8	0	0
	4	0	0
	2	1,629	3,258
	1	6	6
Total words transmitted per second			37,744

Table 2-7. ASB Avionics System Mass Memory Requirements

	Memory Words (16 Bit)	
Primary Program (including On-Line Back-up)	111,000	
Spare Primary	55,500	
Growth	166,500	
Total Primary		333,000
Back-up Program (75 percent of Primary Program)	83,250	
Spare Backup Program (50 percent)	41,625	
Growth Back-up Program (100 percent)	124,875	
Total Back-up Program		249,750
CITS Avionics Fault Isolation	15,000	
CITS Spare Avionics Fault Isolation (50 percent)	7,500	
CITS Growth Avionics Fault Isolation (100 percent)	22,500	
CITS Non-Avionics Fault Isolation	10,000	
CITS Spare Non-Avionics Fault Isolation (50 percent)	5,000	
CITS Growth Non-Avionics Fault Isolation (100 percent)	15,000	
Total CITS Fault Isolation		75,000
Mission Data	25,000	
Spare Mission Data (50 percent)	12,500	
Growth Mission Data (100 percent)	37,500	
Total Mission		75,000
Total Bulk Memory Requirements		732,750 (16 bit words)

1. Primary Program
2. Back-up Program
3. CITS Avionics Fault Isolation
4. CITS Non-Avionics Fault Isolation
5. Mission Data

The primary program contains a duplicate copy of the program in the central computer. The back-up program contains a partial or degraded mode version of the primary program. It is used in the event failures in the central computer result in insufficient capability to perform the primary program. The CITS fault isolation routines are loaded into the computer in the event the CITS function detects a failure and additional routines are required to isolate the failure.

2.4 EXAMPLE OF DETAILED PROCESSING REQUIREMENTS

The detailed data used to derive the processing requirements for the major functions are given in Appendix A. An example of some of these data for the Navigation function is given in this section. Figure 2-6 indicates the primary interconnections of the avionics subsystems with the Navigation function in the central computer. The Navigation function has four subfunctions: IMU Control, Ground Alignment, Navigate, and SRAM Alignment. The interaction of each of these subfunctions with the avionics subsystems and also with the other major processing functions in the central computer is shown in Figure 2-7.

The navigation function was broken down into 15 tasks as shown in Table 2-8. This table indicates the iteration rate, amount of memory in words (without regards to word length), and throughput required in thousands of operations per second for each task. The four subfunctions shown in Figure 2-7 also identify the tasks and execution rate associated with each subfunction.

Table 2-9 is an example of the detailed description of the processing requirements for the tasks. In this table, Tasks 1.1 and 1.2 are broken down into subtasks. The prerequisite tasks to these tasks are also identified as shown in Table 2-9 (Task 9.1 is a prerequisite to both Tasks 1.1 and 1.2). In addition, the last column in Table 2-9 indicates whether these particular tasks contribute to the worst case speed requirements. For the navigation function all Tasks except 1.14 and 1.15 contributed to the worst case speed requirements.

Table 2-10 contains a tabulation of the information transfer required between the tasks. This table is necessary when local vs central processing is considered and separate tasks are performed locally at the subsystem rather than in the central computer.

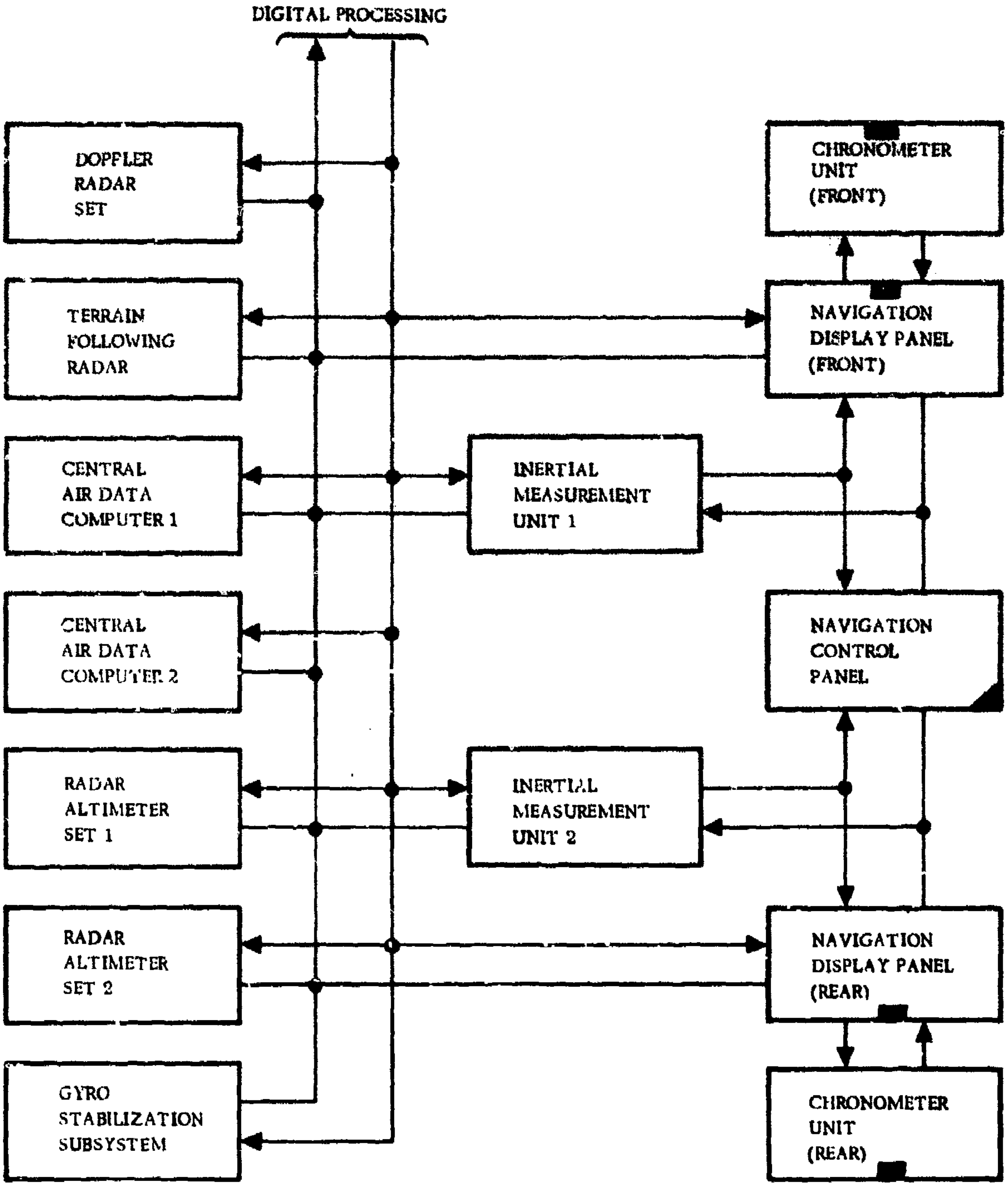
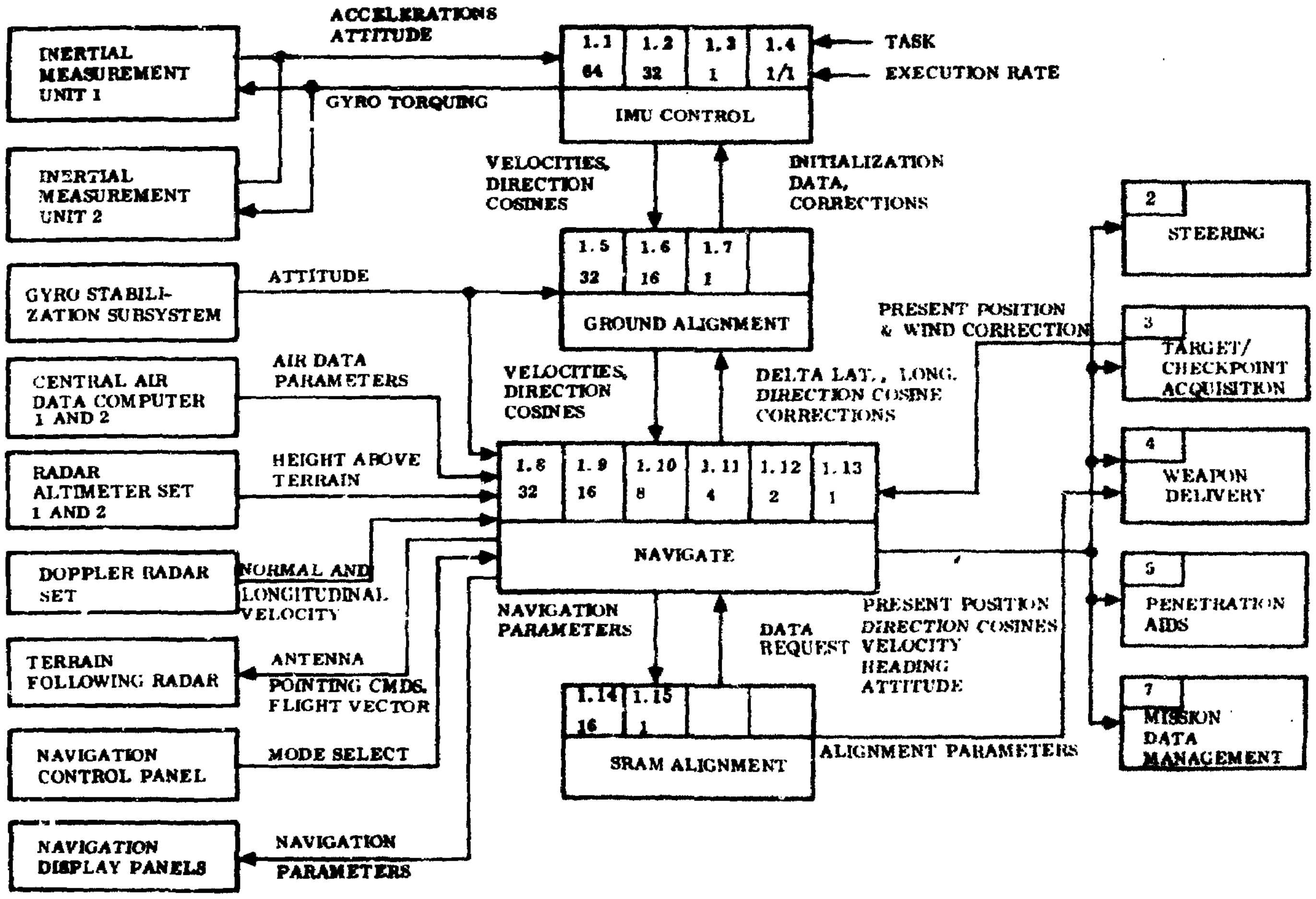

DIGITAL PROCESSING
DOPPLER RADAR SET
TERRAIN FOLLOWING RADAR
CENTRAL AIR DATA COMPUTER 1
CENTRAL AIR DATA COMPUTER 2
RADAR ALTIMETER SET 1
RADAR ALTIMETER SET 2
GYRO STABILIZATION SUBSYSTEM
INERTIAL MEASUREMENT UNIT 1
INERTIAL MEASUREMENT UNIT 2
CHRONOMETER UNIT (FRONT)
NAVIGATION DISPLAY PANEL (FRONT)
NAVIGATION CONTROL PANEL
NAVIGATION DISPLAY PANEL (REAR)
CHRONOMETER UNIT (REAR)

Figure 2-6. Navigation Function Equipment Interface



INERTIAL MEASUREMENT UNIT 1
INERTIAL MEASUREMENT UNIT 2
GYRO STABILIZATION SUBSYSTEM
CENTRAL AIR DATA COMPUTER 1 AND 2
RADAR ALTIMETER SET 1 AND 2
DOPPLER RADAR SET
TERRAIN FOLLOWING RADAR
NAVIGATION CONTROL PANEL
NAVIGATION DISPLAY PANELS
ACCELERATIONS ATTITUDE
GYRO TORQUING
ATTITUDE
AIR DATA PARAMETERS
HEIGHT ABOVE TERRAIN
NORMAL AND LONGITUDINAL VELOCITY
ANTENNA POINTING CMDS. FLIGHT VECTOR
MODE SELECT
NAVIGATION PARAMETERS
1.1 64
1.2 32
1.3 1
1.4 1/1
IMU CONTROL
TASK
EXECUTION RATE
VELOCITIES, DIRECTION COSINES
INITIALIZATION DATA, CORRECTIONS
1.5 32
1.6 16
1.7 1
GROUND ALIGNMENT
VELOCITIES, DIRECTION COSINES
DELTA LAT., LONG. DIRECTION COSINE CORRECTIONS
1.8 32
1.9 16
1.10 8
1.11 4
1.12 2
1.13 1
NAVIGATE
NAVIGATION PARAMETERS
DATA REQUEST
1.14 16
1.15 1
SRAM ALIGNMENT
PRESENT POSITION & WIND CORRECTION
PRESENT POSITION DIRECTION COSINES VELOCITY HEADING ATTITUDE
ALIGNMENT PARAMETERS
2 STEERING
3 TARGET/ CHECKPOINT ACQUISITION
4 WEAPON DELIVERY
5 PENETRATION AIDS
7 MISSION DATA MANAGEMENT

Figure 2-7. Navigation Function Block Diagram

Table 2-8. Navigation Function Processing Requirements Summary

Task	Title	Rate (It/sec)	Memory (words)	Time (KOPS/sec)
1.1	IMU Control - Fast	64	218	18.02
1.2	IMU Control - Mid	32	653	37.65
1.3	IMU Control - Slow	1	394	0.46
1.4	IMU Control - Filter	1	217	0.58
1.5	Ground Alignment - Fast	32	76	2.87
1.6	Ground Alignment - Mid	16	696	13.50
1.7	Ground Alignment - Slow	1	384	0.75
1.8	Navigate - Fast	32	882	32.90
1.9	Navigate - 16/sec	16	240	13.76
1.10	Navigate - 8/sec	8	180	2.88
1.11	Navigate - 4/sec	4	500	1.44
1.12	Navigate - Slow	2	355	0.54
1.13	Navigate - Filter	1/8	7632	0.65
1.14	SRAM Alignment - Fast	16	550	2.50
1.15	SRAM Alignment - Slow	1	240	0.18
			13,217	126.00

Table 2-9. Navigation Function Detail Processing Requirements

Task	Title/Description	Pre-Req	Ops/It	It/Sec	KOPS/Sec	W_c
1.1	IMU Control - Fast 1. IMU Switching 2. Accelerometer Sampling 3. Gyro Torquing					
	Task Total	9.1	281	64	18.02	*
1.2	IMU Control - Mid 1. IMU Switching 2. Incremental Platform Velocities 3. Gravity 4. Coriolis 5. Vertical Velocity 6. Total Platform Velocities 7. Earth Radius 8. Platform Relative Rates 9. Spatial Rates 10. Platform Control Rates 11. Gyro Control Rates 12. Gyro Torque Angle Residual 13. Direction Cosines 14. Platform Azimuth 15. Acceleration Sensitive Drifts 16. Platform Controller Extrapolation 17. Platform Slew 18. Platform Stabilization					
	Task Total	9.1	1176	32	37.65	*

Table 2-10. Navigation Intrafunction Signals

Signal Destination Task Signal Source Task	1.1 IMU Control - Fast	1.2 IMU Control - Mid	1.3 IMU Control - Slow	1.4 IMU Control - Filter	1.5 Ground Alignment - Fast	1.6 Ground Alignment - Mid	1.7 Ground Alignment - Slow	1.8 Navigate - Fast	1.9 Navigate - 16/sec	1.10 Navigate - 8/sec	1.11 Navigate - 4/sec	1.12 Navigate - Slow	1.13 Navigate - Filter	1.14 SRAM Alignment - Fast	1.15 SRAM Alignment - Slow
1.1 IMU Control - Fast				8	12			12							
1.2 IMU Control - Mid				8		18			18						
1.3 IMU Control - Slow				6			10			10					
1.4 IMU Control - Filter	8	8	6												
1.5 Ground Alignment - Fast								72							
1.6 Ground Alignment - Mid									36						
1.7 Ground Alignment - Slow												10			
1.8 Navigate - Fast		2							36	36	36	36	72	48	48
1.9 Navigate - 16/sec								36		36	36	36	36	36	36
1.10 Navigate - 8/sec		4						12	12		12	12	12	12	12
1.11 Navigate - 4/sec		13						10	10	10		10	10	10	10
1.12 Navigate - Slow								10	10	10	10		10	10	10
1.13 Navigate - Filter								72	36	12	10	10			
1.14 SRAM Alignment - Fast															
1.15 SRAM Alignment - Slow															

3. MODULE DEFINITION

3.1 BURROUGHS COMPUTER DESCRIPTION

The Burroughs computer concept has been referred to by various acronyms in recent reports (Ref 1, 2, 3, 4, 5) such as the Interpreter Based System, Multiprocessor, and Aerospace Multiprocessor; in addition, the title of this study uses the acronym, avionics processor - controller. The term Burroughs multiprocessor or simply the multiprocessor will be used in place of these acronyms in this report.

A block diagram indicating the general structure of the Burroughs multiprocessor is given in Figure 3-1. The basic modules or building blocks of the multiprocessor are:

1. Interpreters - Processing Elements consisting of arithmetic logic and alterable microprogram controls
2. SWI (Switch Interlock Unit) - Interconnection logic to allow interpreters to communicate with memories and devices
3. Memories - Storage elements for programs and data
4. Devices - interface elements between peripherals and the SWI
5. PSU (Port Select Unit) - May be used in place of the SWI for single interpreter systems

The Burroughs multiprocessor emphasizes two concepts (a) building block structure and (b) variable machine architecture achieved through microprogramming.

The basic building blocks listed above allow multiprocessors with different numbers of modules to be constructed to meet varying computational requirements. The multiprocessor designed for the Air Force allowed up to five Interpreters, eight Memories, and eight Devices.

Variable machine architecture is possible with the Burroughs multiprocessor by reloading the microprogram memory with routines. For example it is possible to (a) emulate existing computers, (b) perform higher order language processing, and (c) process a problem optimized instruction set. Further, these could be performed concurrently in a multiprocessing manner.

The computer can operate as a true multiprocessor since any interpreter may access any memory or device module and multiple interpreters may be used simultaneously to process a computational task. Through the flexibility offered by variable machine architecture the interpreter can function as a CPU, as an I/O Processor, or as a device controller.

The Burroughs multiprocessor modules will be described below. Appendix B contains additional details on the modules and is a collection of extracts from references 1 through 5.

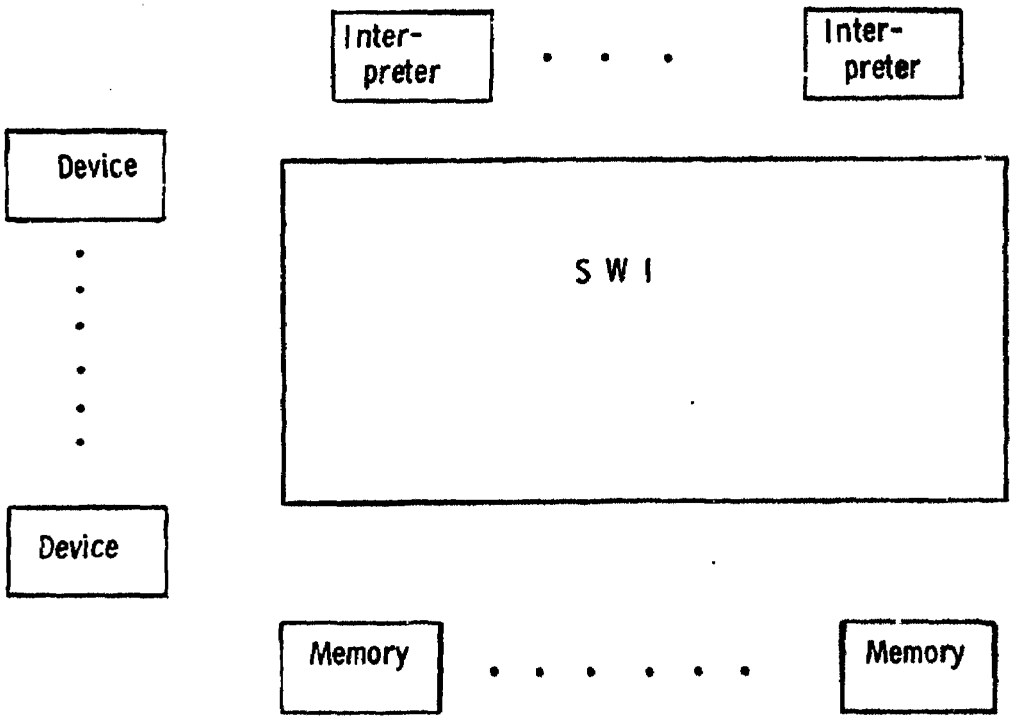


Figure 3-1. Burroughs Multiprocessor

3.2 INTERPRETER DESCRIPTION

3.2.1 General

The interpreter's functions are to:

1. Contain the microprogram memory.
2. Provide the timing and control for sequencing and controlling according to the microprogram memory.
3. Control the communication with external devices and memories.
4. Perform the logical and arithmetic operations required.

In order to accomplish these in a flexible manner Burroughs has defined a modular approach with the Interpreter consisting of submodules as follows: (see Figure 3-2):

1. Logic Unit - The circuitry associated with the arithmetic, shifting, and logic functions are contained in the Logic Unit. The data word length is expandible from 8 to 64 bits in 8 bit increments.
2. Control Unit - The Control Unit contains registers for conditional control and logic commands.
3. Memory Control Unit - The Memory Control Unit provides registers and control for memory (interpreter and main memory) addressing.
4. Microprogram Memory - This unit provides storage for the microprogram sequences. The unit could be implemented with ROM or RAM devices.
5. Nanomemory - The microcontrols for an Interpreter are supplied by the 54 bit wide Nanomemory. Most likely implementation of this is with the use of ROM. The particular nanoword is selected by the MPM word using the contained memory address.

3.2.2 Logic Unit (LU)

Figure 3-3 contains a detailed description of the data and control flow in the interpreter and Table 3-1 identifies the control provided by the 54 bits in the nanomemory word. Reference to Figure 3-3 and Table 3-1 will aid in following the interpreter description given below.

One Logic Unit for each 8 bits of data word is required for each interpreter. The LU is composed of: the three A registers, a B register, an MIR register, adder, and barrel switch logic.

Registers A1, A2, and A3 are functionally identical. Each temporarily stores data and serves as a primary input to the adder. Any of the A registers can be loaded with the output of the barrel switch.

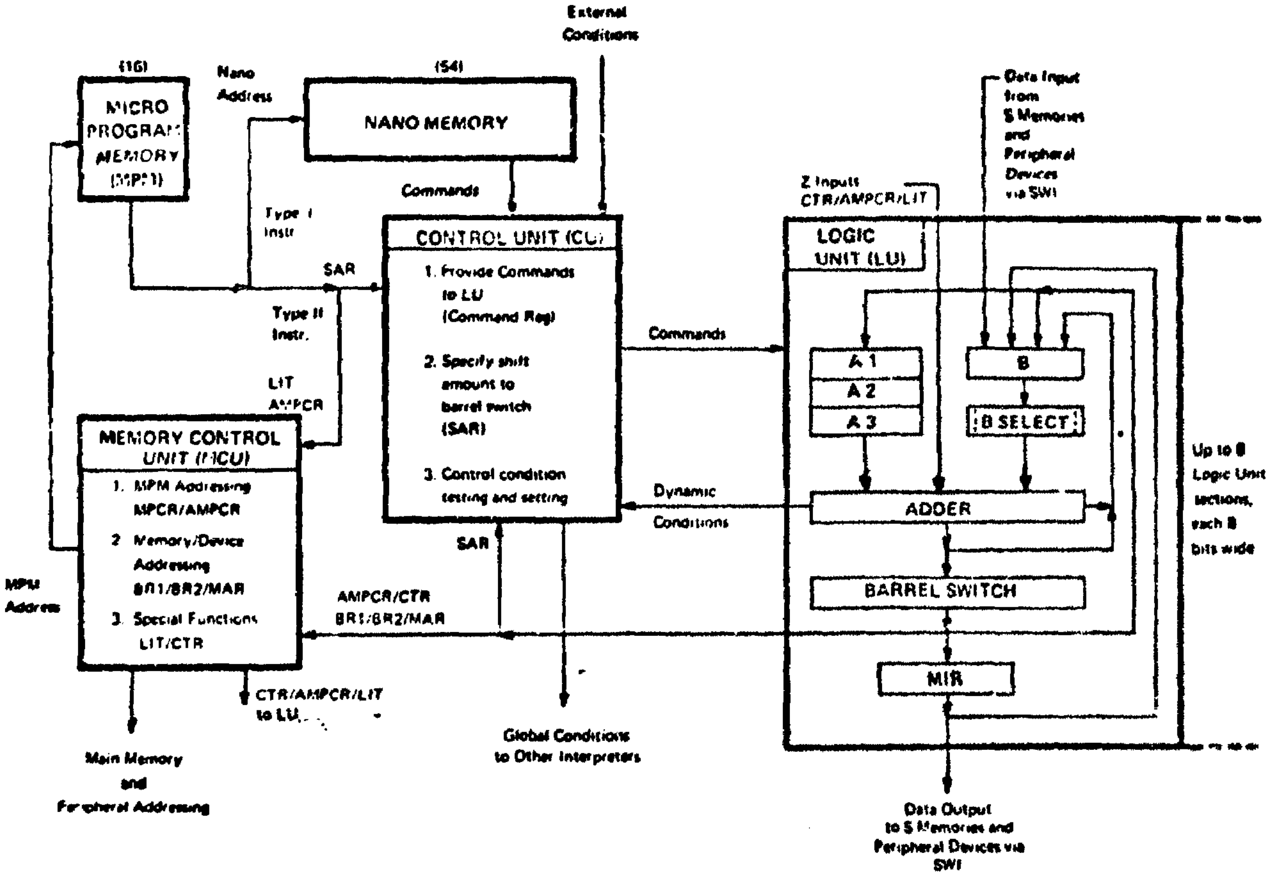
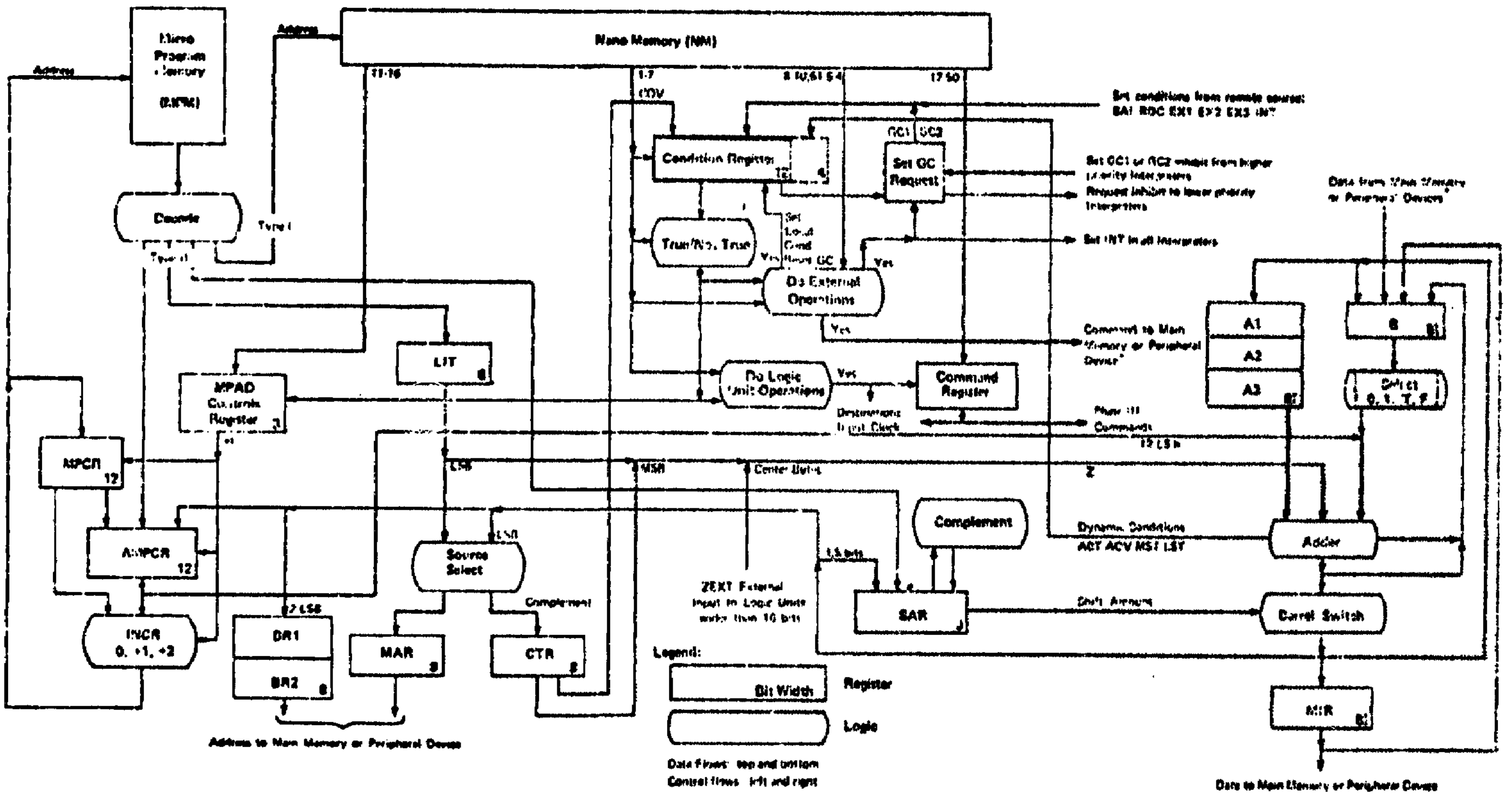

External Conditions
(16)
MICRO PROGRAM MEMORY (MPM)
Nano Address
(54)
NANO MEMORY
Commands
Type I Instr
SAR
Type II Instr.
LIT AMPCR
CONTROL UNIT (CU)
1. Provide Commands to LU (Command Reg)
2. Specify shift amount to barrel switch (SAR)
3. Control condition testing and setting
SAR
MEMORY CONTROL UNIT (MCU)
1. MPM Addressing MPCR/AMPCR
2. Memory/Device Addressing BR1/BR2/MAR
3. Special Functions LIT/CTR
MPM Address
AMPCR/CTR BR1/BR2/MAR
CTR/AMPCR/LIT to LU
Main Memory and Peripheral Addressing
Global Conditions to Other Interpreters
Commands
Dynamic Conditions
Z Inputs CTR/AMPCR/LIT
Data Input from S Memories and Peripheral Devices via SWI
LOGIC UNIT (LU)
A 1
A 2
A 3
B
B SELECT
ADDER
BARREL SWITCH
MIR
Up to 8 Logic Unit sections, each 8 bits wide
Data Output to S Memories and Peripheral Devices via SWI

Figure 3-2. Interpreter Block Diagram


Micro Program Memory (MPM)
Address
Address
Nano Memory (NM)
Decode
Type I
Type II
11-16
1-7
COV
8-10,61-64
17-50
Condition Register
16
4
GC1
GC2
Set GC Request
Set conditions from remote source: SAI RDC EX1 EX2 EX3 INT
Set GC1 or GC2 inhibit from higher priority interpreters
Request inhibit to lower priority interpreters
Set INT in all interpreters
True/Not True
Set Local Cond from GC
Yes
Do External Operations
Yes
Command to Main Memory or Peripheral Device*
Do Logic Unit Operations
Yes
Destination Input Clock
Command Register
Phase III Commands
LIT
8
MPAD Controls Register
3
+1
MPCR
12
AMPCR
12
INCR 0, +1, +2
LSB
MSB
Center Bytes
12 LSb
Z
LSB
Source Select
Complement
2 LSB
BR1
BR2
8
MAR
8
CTR
8
Address to Main Memory or Peripheral Device
ZEXT External Input to Logic Unit wider than 16 bits
LS bits
Complement
SAR
J
Shift Amount
Dynamic Conditions ACT ACV MST LST
A1
A2
A3
B
Z
Data from Main Memory or Peripheral Devices*
Shift 0, 1, T, F
Adder
Barrel Switch
MIR
Data to Main Memory or Peripheral Device
Legend:
Bit Width
Register
Logic
Data Flows: top and bottom
Control flows: left and right

*SUCCESSOR

	Select	Incr.	
WAIT	MPCR	0	
STEP	MPCR	1	Provided for Type II and unless otherwise specified, Type I
SKIP	MPCR	2	
CALL	AMPCR	1	Also saves MPCR contents in AMPCR (for return)
JUMP	AMPCR	1	
RETN	AMPCR	2	
EXEC	AMPCR	1	Inhibits loading of MPCR
SAVE	MPCR	1	Saves MPCR contents in AMPCR (for looping)

REGISTER NAMES

MPCR	Microprogram Count Register
AMPCR	Alternate Microprogram Count Register
BR1	Base Register 1
BR2	Base Register 2
MAR	Memory Address Register
CTR	Counter
SAR	Shift Amount Register
A1	A1 Register
A2	A2 Register
A3	A3 Register

NOTES

*Communication with Main Memory or Peripheral Devices is via Port Select Unit or Switching Interlock

Abbreviations:	
LSB	= Least Significant Byte
MSB	= Most Significant Byte
2 LSB	= Second LSB
12 LSb	= 12 Least Significant bits
Z	= Byte width of Logic Unit
J	= Least integer ≥ $3 + \log_2 Z$

Figure 3-3. Interpreter Data and Control Flow

Table 3-1. Detailed Nanobit Assignment

[illegible] MICROCONTROLS

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	MAR					.	.	.	.	.	.	.	.
0	1	SAR					LIT							
0	0	1	-	-	AMPCR									
0	0	0	1	-	-	-	LIT							
0	0	0	0	-	-	-	NANO ADDRESS							

–Lexsix (Assembled as zeros)
Pseudo Operation to Card Reader Loader which terminates loading of MPM and Nano, goes to MPM location ··· ro and begins execution.

NANO CONTROLS

Parentheses surround optional logic units, provided by default.

[1 2 3 4] Condition Tested
Result is boolean and

1	2	3	4	
0	0	0	0	(GC1)
0	0	0	1	GC2
0	0	1	0	LC1
0	0	1	1	LC2
0	1	0	0	MST
0	1	0	1	LST
0	1	1	0	ABT
0	1	1	1	AOV
1	0	0	0	COV
1	0	0	1	[illegible]
1	0	1	0	RDL
1	0	1	1	LC3
1	1	0	0	EXT
1	1	0	1	INT
1	1	1	0	ENT
1	1	1	1	[illegible]

*See Adder Operations for explanation

[5] PT Condition Value

5	
0	NOT SC ← (R)
1	SC ← R

[6] Logic Unit Conditional

6	
0	Do Unconditionally
1	Do Conditionally if SC

[7] Ext Op (MDOP & SAR) Conditional

7	
0	Do Unconditionally
1	Do Conditionally if SC

[8 9 10] Condition Adjust -- CAJ

8	9	10	
0	0	0	--
0	0	1	SET LC2
0	1	0	SET GC2
0	1	1	RESET GC
1	0	0	SET INT
1	0	1	SET LC3
1	1	0	SET GC1
1	1	1	SET LC1

[11 12 13] Successor [14 15 16]

11	12	13	Then Part Used if SC=1	14	15	16	Else Part Used if SC=0
0	0	0	WAIT	0	0	0	
0	0	1	STEP	0	0	1	
0	1	0	SAVE	0	1	0	
0	1	1	SKIP	0	1	1	
1	0	0	JUMP	1	0	0	
1	0	1	EXEC	1	0	1	
1	1	0	CALL	1	1	0	
1	1	1	RETN	1	1	1	

[17 18] Adder A Input

17	18	
0	0	(0)
0	1	A1
1	0	A2
1	1	A3

[19 20 21 22 23 24] Adder B Input

19	20	21	22	23	24	
0	0	-	-	-	-	B0--
0	1	-	-	-	-	BF--
1	0	-	-	-	-	BT--
1	1	-	-	-	-	B1--
-	-	0	0	-	-	B-0-
-		0	1	-	-	B-F-
-	-	1	0	-	-	B-T-
-	-	1	1	-	-	B-1-
-	-	-		0	0	B--0
	-	-		0	1	B--F
-	-	-	-	1	0	B--T
-	-	-	-	1	1	B--1

[25 26] Adder Z Input

25	26		
0	0	(0)	
0	1	CTR	MS byte
1	0	LIT	LS byte
1	1	AMPCR	Middle bytes

[27] Inhibit Carries into Bytes

27		
0	--	Allow
1	IC	Inhibit

[28 29 30 31] Adder Operation

28	29	30	31	Adder Operation	ABT+1 (Condition 6) if all adder outputs are
0	0	0	0	A+Z+1	0's
0	0	0	1	A+Z	1's
0	0	1	0	(A+B)	1's
0	0	1	1	A+B+1	0's
0	1	0	0	$\bar{B}$(A XOR Z)	1's
0	1	0	1	$\bar{A}$Z ∨ BZ	1's
0	1	1	0	A$\bar{B}$ ∨ $\bar{A}$B	1's
0	1	1	1	A$\bar{B}$	1's
1	0	0	0	B+Z+1	0's
1	0	0	1	B+Z	1's
1	0	1	0	$\bar{A}$B	0's
1	0	1	1	A ∨ B	1's
1	1	0	0	A ∨ B ∨ Z	0's
1	1	0	1	AZ ∨ BZ	0's
1	1	1	0	(A ∨ B) EQ Z	0's
1	1	1	1	$\bar{A}$ ∨ B	0's

For Explanation of TRANSLANG Adder Operations, see Appendix II.

[32 33] Shift Type Selections for BSW

32	33		
0	0	--	No Shift
0	1	R	Right End Off
1	0	L	Left End Off
1	1	C	Right Circular

[34 35 36] A Register Input from BSW

34	35	36		
0	0	0	--	No Change
1	-	-	A1	
-	1	-	A2	
-	-	1	A3	

[37* 38 39 40] B Register Input Select

37*	38	39	40		
-	0	0	0	--	No Change
0	0	0	1	BAD	Adder (unshifted)
0	0	1	0	BSW	BSW
0	0	1	1	BBA	BSW ∨ Adder
0	1	0	0	BC4	Comp 4 Bit Carries
0	1	0	1	BC8	Comp 8 Bit Carries
0	1	1	0	BBE	BSW ∨ External
0	1	1	1	BEX	External
1	0	0	1	BAM	Adder ∨ MIR
1	-	1	-	BBM	BSW ∨ MIR
1	0	1	1	BBAM	BSW ∨ Adder ∨ MIR
1	1	0	0	B4M	Comp 4 Bit Carries ∨ MIR
1	1	0	1	B8M	Comp 8 Bit Carries ∨ MIR
1	1	1	1	BMI	MIR

*Bit 37 true from the command register will clear the input shift register from the switch interlock.

[41] MIR Input from BSW

41		
0	--	No Change
1	MIR	

[42] AMPCR Input from BSW

42		
0	--	No Change
1	AMPCR	

[43 44 45 46] Mem Dev Address Input

43	44	45	46		
0	0	0	-	--	No Change
-	-	1	0	LMAR	From LIT
-	-	1	1	MAR	From BSW
-	1	-	-	BR2	From BSW
-	1	1	1	MAR2	From BSW
1	-	-	-	BR1	From BSW
1	-	1	1	MAR1	From BSW

[46 47 48] Counter Input

46	47	48		
-	0	0	--	No Change
0	0	1	LCTR	From LIT
1	0	1	CTR	From BSW*
-	1	-	INC	+1

*Ones Complement

[49 50] SAR Input

49	50		
0	0	--	No Change
0	1	CSAR	Complement
1	0	SAR	From BSW

[51 52 53 54] Mem Dev Op-MDOP

51	52	53	54		
0	0	0	0	--	No Change
0	0	0	1	LDM	Load MPM*
0	0	1	0	MR1	Mem Read (BR1)
0	0	1	1	MR2	Mem Read (BR2)
0	1	0	0	LDN	Load Nanomemory*
0	1	1	0	MW1	Mem Write (BR1)
0	1	1	1	MW2	Mem Write (BR2)
1	0	0	0	DL1	Dev. Lock (BR1)
1	0	0	1	DL2	Dev. Lock (BR2)
1	0	1	0	DR1	Dev. Read (BR1)
1	0	1	1	DR2	Dev. Read (BR2)
1	1	0	0	DU1	Dev. Unlock (BR1)
1	1	0	1	DU2	Dev. Unlock (BR2)
1	1	1	0	DW1	Dev. Write (BR1)
1	1	1	1	DW2	Dev. Write (BR2)

*From least significant 16 bits of MIR

The B register is a primary external interface (from the Switch Interlock). It serves as the second input to the adder and may be loaded with any of the following:

1. The barrel switch output.

2. The adder output.

3. The data from the Switch Interlock.

4. The MIR output.

5. The carry complements (from the adder) of 4- or 8-bit groups with selected zeroes (for use in decimal arithmetic or character processing).

6. The barrel switch output ORed with 2, 3, or 4 above.

The MIR buffers information being written to main memory or to a peripheral device. It is loaded from the barrel switch output and its output is sent to the Switch Interlock, or to the B register.

Inputs to the adder are from selection gates which allow various combinations of the A, B and Z inputs. The Z input is an external input to the LU and can be:

1. The 8-bit output of the counter of the MCU into the most significant 8 bits with all other bits being ZEROs.

2. The 8-bit output of the literal register of the MCU into the least significant 8 bits with all other bits being ZEROs.

3. The 12-bit output of the alternate microprogram count register (AMPCR) right justified into the middle 16 bits and the (wired) Interpreter number right justified in the remaining four bits of the middle 16 bits. All other bits are zeros.

4. All ZEROs.

Using various combinations of inputs to the selection gates, any two of the three inputs can be added together, or can be added together with an additional ONE added to the least significant bit. Also, all binary Boolean operations between the A and B and between the B and Z adder inputs and most of the binary Boolean operations between the A and Z adder inputs can be done.

The barrel switch is a matrix of gates that shifts a parallel input data word any number of places to the left or right, either end-off or end-around, in one clock time.

The output of the barrel switch is sent to:

1. The A registers (A1, A2, A3).

2. The B register.

3. Memory Information Register (MIR).

4. Least significant 16 bits to MCU (registers BR1, BR2, MAR, AMPCR, LIT, CTR).

5. Least significant five bits to shift amount register (SAR) in the CU.

3.2.3 Control Unit (CU)

Major sections of the CU are: the shift amount register (SAR), the condition register, part of the control register (CR), the MPM content decoding, and the clock control.

The condition register section of the CU performs four major functions:

1. Stores 12 resettable condition bits in the condition registers. The 12 bits of the condition register are used as error indicators, interrupts, status indicators and lockout indicators.

2. Selects 1 of 16 condition bits (12 from the register and 4 generated during the present clock time in the Logic Unit) for use in performing conditional operations.

3. Decodes bits from the Nanomemory for resetting, setting, or requesting the setting of certain bits in the condition register.

4. Resolves priority between Interpreters in the setting of global condition (GC) bits.

3.2.4 Memory Control Unit (MCU)

This unit has three major sections:

1. The microprogram address section contains the microprogram count register (MPCR), the alternate microprogram count register (AMPCR), the incrementer, the microprogram address control register, and associated control logic. The output of the incrementer addresses the MPM for the sequencing of the microinstructions. The AMPCR contents are also used as one of the Z inputs to the adder in the LU.

2. The memory/device address section contains the main memory address register (MAR), base registers one and two (BR1, BR2), the base register output selection gates, and the associated control logic.

3. The Z register section contains registers which are two of the Z inputs to the LU adder: a loadable counter (CTR), the literal register (LIT), selection gates for the input to the memory address register and the loadable counter and their associated control logic.

3.2.5 Interpreter Operation

A unique feature of the Interpreter Based System is the utilization of stored logic in M and N memories and uncommitted hardware logic to form firmware control that is exercised to a more primitive logic level than in conventional microprogrammed central processors. During each clock period, a 16 bit microinstruction is read from the MPM. The first four bits of this microinstruction indicate which of two types of instructions it is. If it is a Type I instruction, the remaining bits of the MPM word specify a Nanomemory address to be accessed. The Nanomemory is then initiated and its output, a set of 54 bits, provides the control functions as indicated in Table 3-1.

If the microinstruction is Type II, the remaining bits of the MPM word are stored into one or two registers: namely, the SAR, LIT, SAR and LIT, or the AMPCR. The determination of which registers are to be loaded is specified by the first four bits of the MPM word. The Nanomemory is not accessed during a Type II operation.

Each Type I microinstruction has two parts (or phases). The first fetches the instruction from the MPM and Nanomemory and the second executes the fetched instruction. Figure 3-4 illustrates these two basic phases of each Type I microinstruction.

The fetch phase involves: MPM accessing, Nanomemory accessing, condition testing, selection of controls for the next instruction (successor) address computation, and, in parallel, loading the control register for the execution of the microinstruction. A fetch phase occurs for every Type I microinstruction and requires one clock time. Since it always overlaps the execution phase of a prior Type I microinstruction, the performance of each microinstruction requires effectively one clock interval.

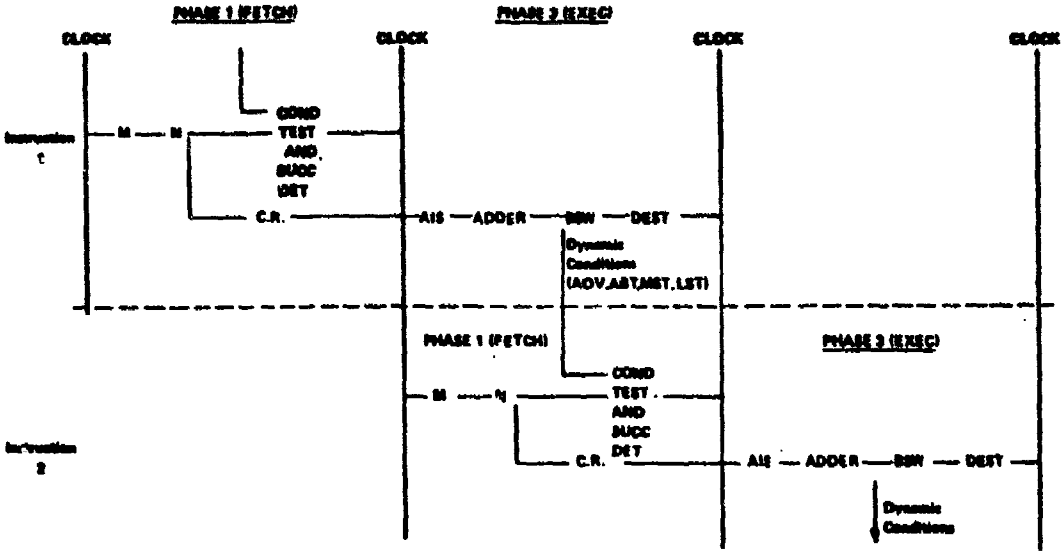


Figure 3-4. Timing Analysis Type I Instructions

The execution phase also required one clock time and always overlaps the fetch phase of the next Type I instruction. The control signals for the execution phase are from the output of the control register and have two parts: signals specifying the logic unit operation (adder input selection, adder function, barrel switch shifting, etc.) and signals specifying the destination register(s) loading (i.e. clock enables). The completion of the execution phase (i.e. the destination register(s) loading), may be delayed or suspended for one or more clock times. This suspended execution phase can occur for three primary reasons. The first and most frequent occurrence is when the next instruction from the MPM is a Type II instruction. The second reason for the occurrence of a suspended execution phase is due to the existence of conditional logic unit operations. The other reason for a suspended execution phase is for use during the loading of the MPM and Nanomemory.

The sequencing of Type I microprogram instructions is controlled by information contained in the nanomemory word which provides three true and three false condition bits for selection of the successor Type I microinstruction. The three selected bits (True/False condition) provide eight possible successor commands as listed in Table 3-2.

Table 3-2. Microprogram Memory Addressing

Successor Command	Successor M-instruction Address	Next Content of MPCR will be	Next Content of AMPCR will be
WAIT	MPCR	MPCR	*
STEP	MPCR+1	MPCR+1	*
SKIP	MPCR+2	MPCR+2	*
SAVE	MPCR+1	MPCR+1	MPCR
CALL	AMPCR+1	AMPCR+1	MPCR
EXEC	AMPCR+1	MPCR *	*
JUMP	AMPCR+1	AMPCR+1	*
RETN	AMPCR+2	AMPCR+2	*

* Not changed by successor specification

3.2.6 Multiprocessing Features

The SWI module is the primary hardware feature in the Burroughs multiprocessor that allows the computer to operate as a multiprocessor. There are two additional hardware features included in the interpreter that aid in multiprocessing operation: the global condition bits and the interrupt bit. The global condition bits are two of the 16 testable condition bits in each interpreter. Each bit can be set in only one interpreter at a time and must be programmatically reset. An interpreter nanoinstruction containing the "Set Global Condition Bit" operation will set the specified global condition bit in that interpretor only if that bit is not set in any interpreter and no other higher (wired) priority interpreter is requesting the same bit to be set in its own interpreter. The global condition bits allow a multiprocessing executive to be implemented that requires tables in main memory to be locked such that only one interpreter may be modifying data in these tables at any one time.

One more of the testable condition bits in each interpreter is wired to provide an additional inter-interpreter signal. This bit is called the interrupt bit and is simultaneously set in all interpreters by an operation originating from any interpreter. This bit is reset in an interpreter when tested in that interpreter.

3.3 SWITCH INTERLOCK (SWI) DESCRIPTION

3.3.1 SWI Modules

The Switch Interlock functions are to:

1. Provide the interconnection of the interpreters with the memories and devices.

2. Provide the priority for the interpreters in the selection of devices and memories.

Connection between Interpreters and devices is by reservation with the Interpreter having exclusive use of the (locked) device until specifically released. Connection with a memory module is for the duration of a single data word exchange, but is maintained until some other module is requested or some other Interpreter requests that module.

In any such system it is desirable to keep the wires and logic in the crosspoints to a minimum, while still maintaining a specified transfer rate. One way of achieving this is by serial transmission of several partial words in parallel through the crosspoints. The Switch Interlock for the Burroughs Multiprocessor handles up to five Interpreters, eight memories and eight devices. The transmission paths through the Switch Interlock break the 32-bit data word into 4 - 8 bit bytes.

The SWI is mechanized with five modules; a block diagram indicating the structure of the SWI is given in Figure 3-5. This diagram also shows the internal and external interface of the SWI. The five modules are:

1. Memory Device Control (MDC) - This unit, shown in Figure 3-6 decodes the nanomemory bits and generates the signals for controlling the other SWI modules. The MDC also contains the counter and logic to indicate to its interpreter, data acceptance and transfer completion. There is one MDC per interpreter.

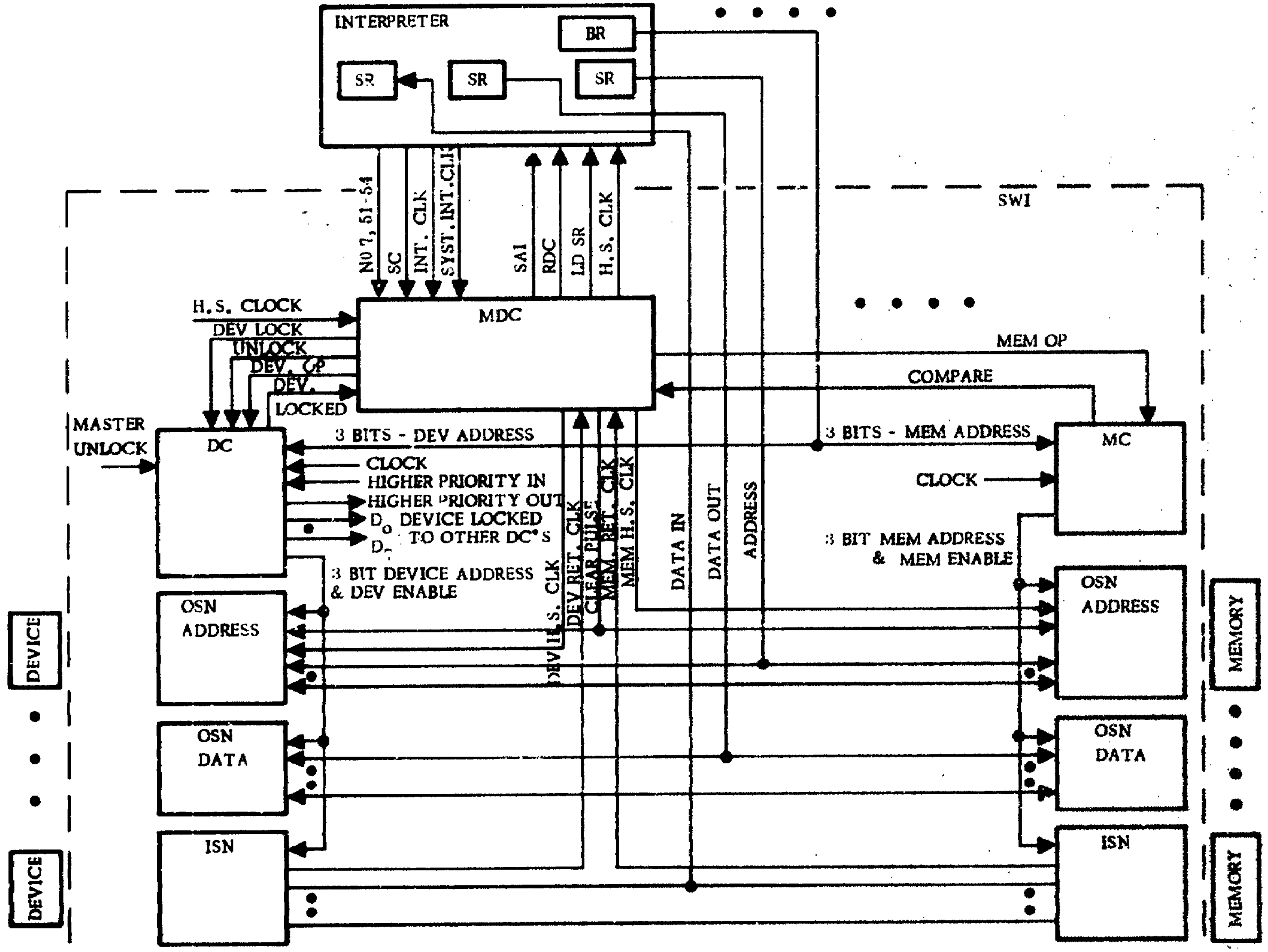


Figure 3-5. SWI Interface Diagram

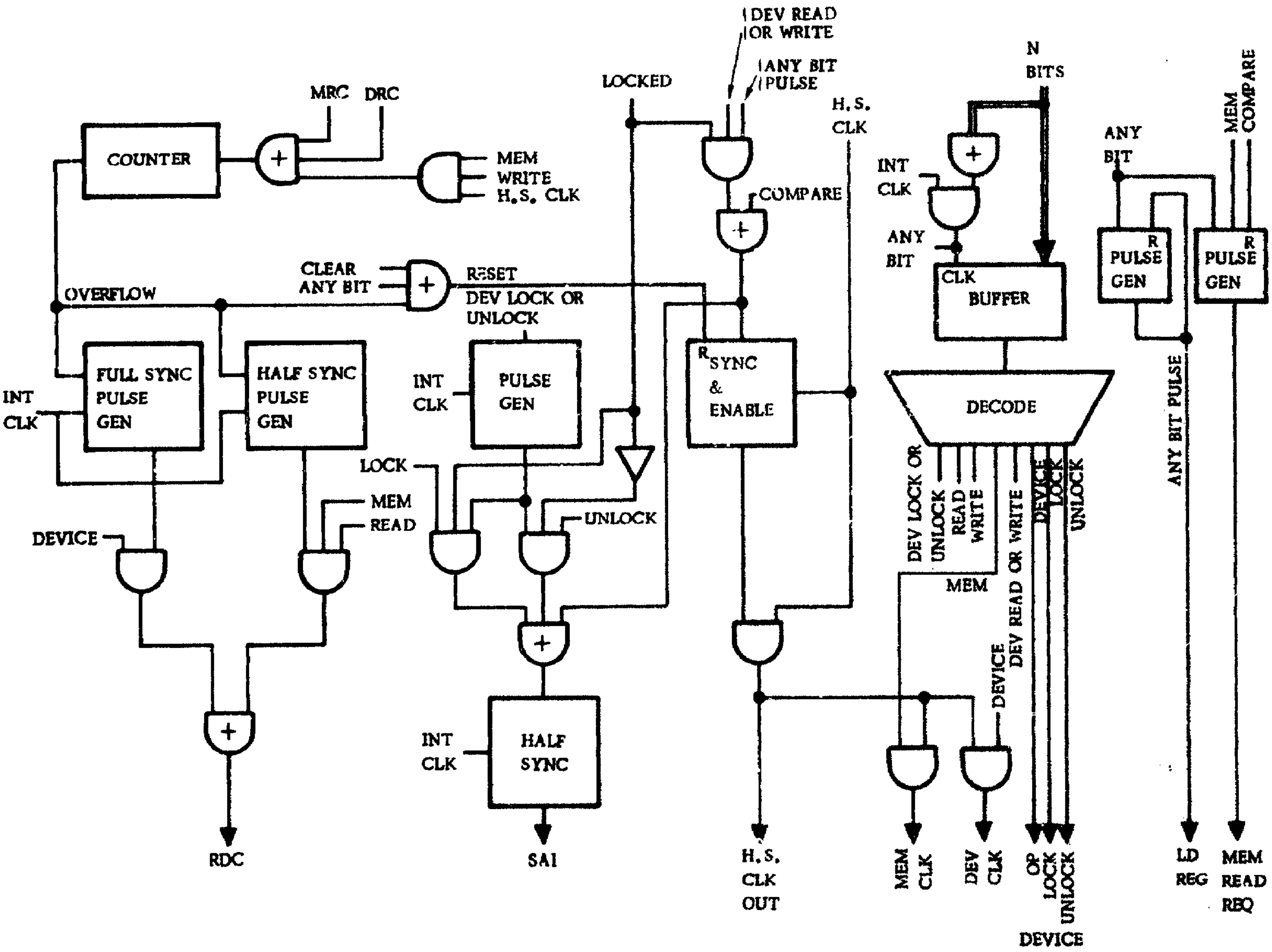


Figure 3-6. MDC Block Diagram

2. Device Control (DC) - The DC resolves conflicts between Interpreters trying to look to a device and checks the lock status of any Interpreter attempting a device operation. The DC is shown in Figure 3-7, it receives requests for device operations and lock/unlock requests through the MDC. It responds by sending status signals to the MDC and control signals to the Input and Output Switch Network modules. The DC module as mechanized in the Burroughs multiprocessor provides device control for up to three interpreters. A system wi th five interpreters will use two DC modules.

3. Memory Control (MC) - The MC resolves conflicts between Interpreters requesting the use of the same memory module and maintains an established connection after completion of the operation until some other Interpreter requests that memory module. Figure 3-8 contains a diagram indicating a typical interpreter stage in the memory control module. This stage receives requests from the MDC and a 3 bit memory module address from the interpreter. The lower section of Figure 3-8 shows the memory request and memory busy bus that connects to the priority logic for memory request control. The Burroughs mechanization of the MC uses two modules MC0 and MC1. MC0 contains three stages as shown in Figure 3-8 to provide memory control for three interpreters. MC1 contains two stages and the memory busy flip-flops.

4. Input Switch Network (ISN) - The ISN returns data from addressed devices or memory modules to the requesting interpreter (i.e., the ISN is a "Multiplexer"). As seen in Figure 3-9 the ISN module provides selection for five interpreters to up to eight memories or eight devices. The ISN provides a path for 10 bits per interpreter. This path is used to provide eight data bits and a return clock, one bit is unused. The ISN module mechanized by Burroughs actually consists of two submodules, each submodule provides for 4 data bits and 1 clock bit from up to eight memories or devices to up to five interpreters. The ISN is therefore modular in terms of 4 bit bytes. The ISN is under the control of the MC or DC module.

5. Output Switch Network (OSN) - The OSN sends data, address, clock, and control from Interpreters to addressed devices or memory modules (i.e., the OSN is a "demultiplexer"). This unit is actually mechanized as two different modules. Figure 3-10 shows the OSN for address output. This unit handles 4-address and 2-clock bits for five interpreters to up to eight memories on devices. The address OSN is actually mechanized from two identical submodules that provide two address bits and one clock bit each. In the Burroughs multiprocessor, the address OSN uses four address and one clock bit leaving one clock bit unused.

   The data output OSN is shown in Figure 3-11. This unit provides eight bits output to up to eight memories or devices from five interpreters.

3.3.2 Switch Interlock Operation and Timing

Controls from the Interpreter (Nanobits 51-54) are strobed into the mem/dev operation register of the MDC if either the Type I microinstruction is unconditional or the selected condition is true. Controls derived from the output of this register will next load the output shift registers of the interpreter and generate one of three types of signals, depending upon the operation to be performed. Each of these types of signals will be explained.

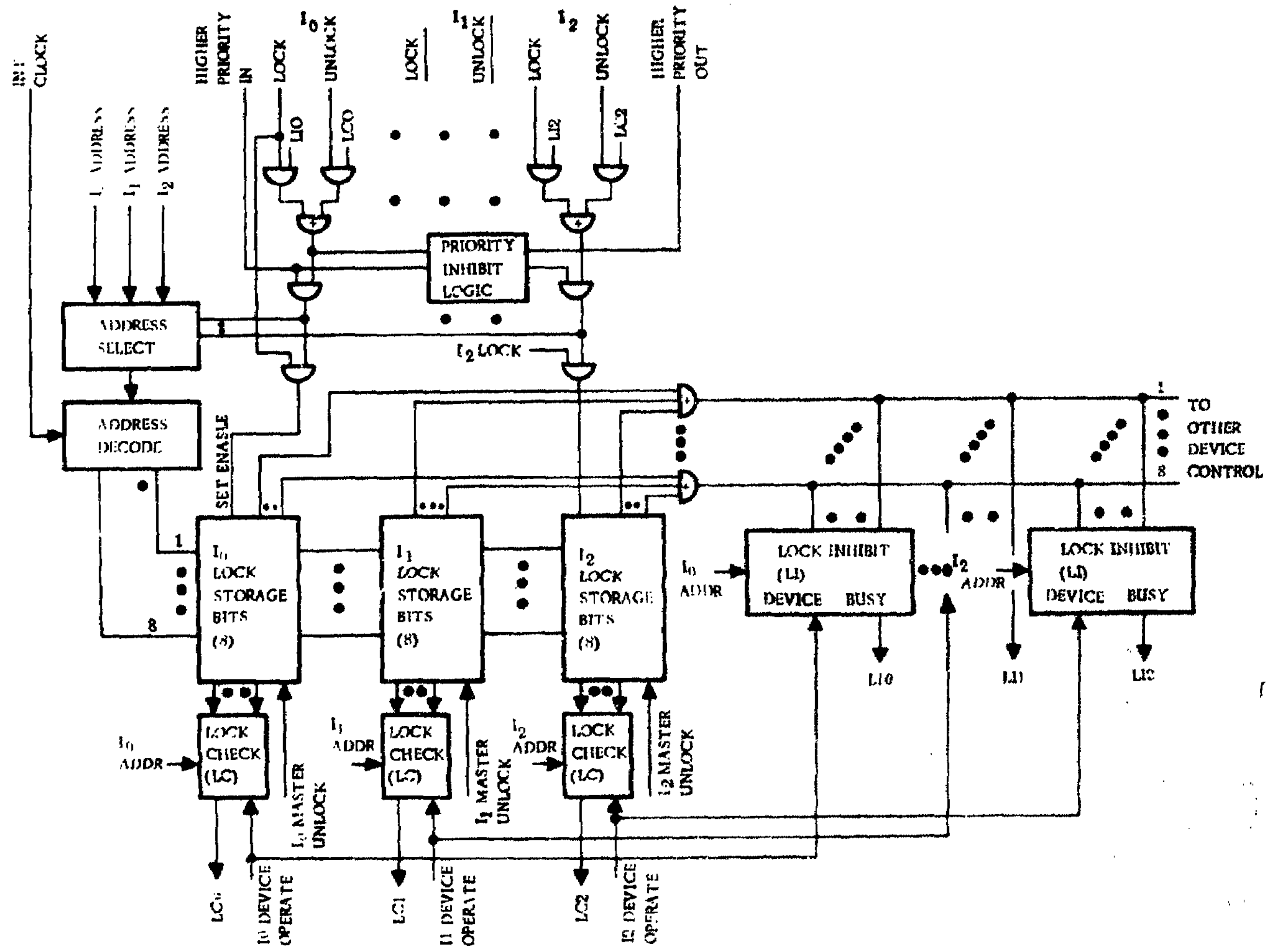


Figure 3-7. 3-Channel Device Control Block Diagram

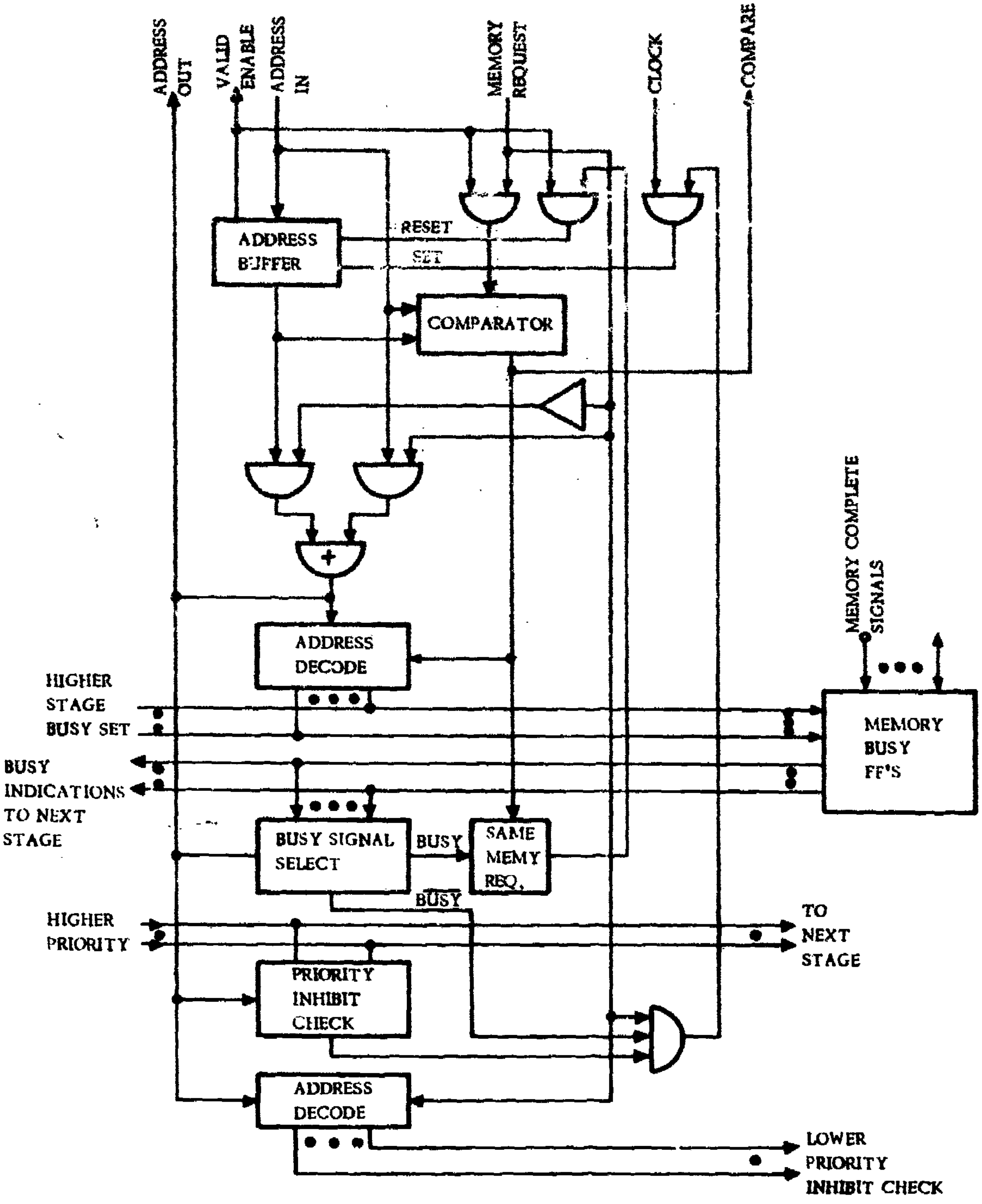


Figure 3-8. Typical Stage - Memory Control

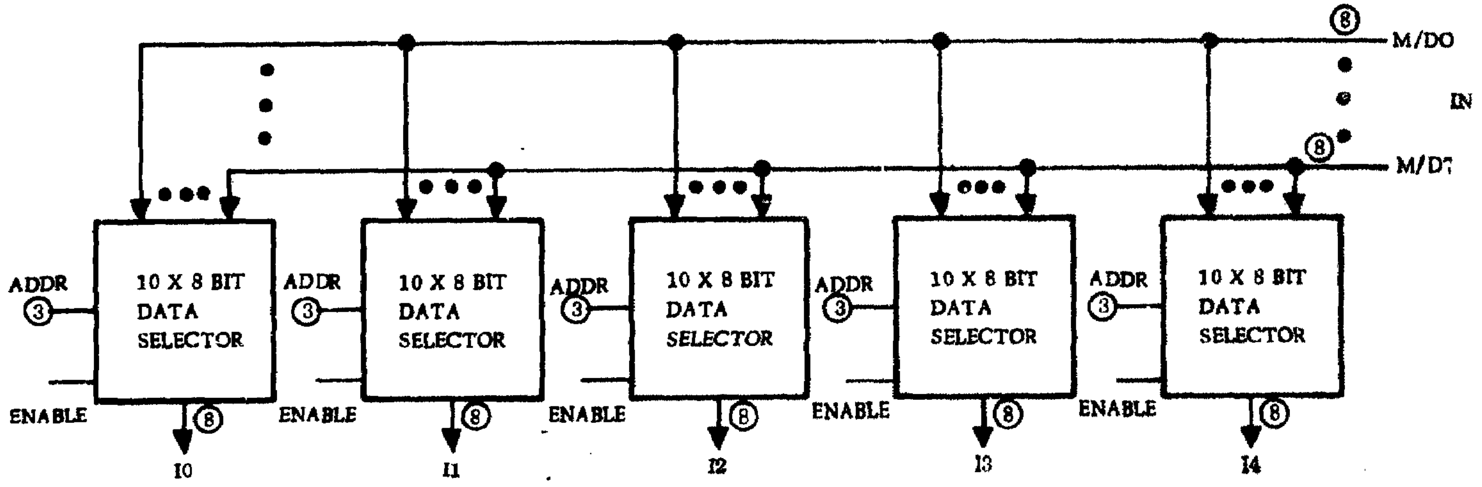


Figure 3-9. 5-Channel Input Switch Network Block Diagram

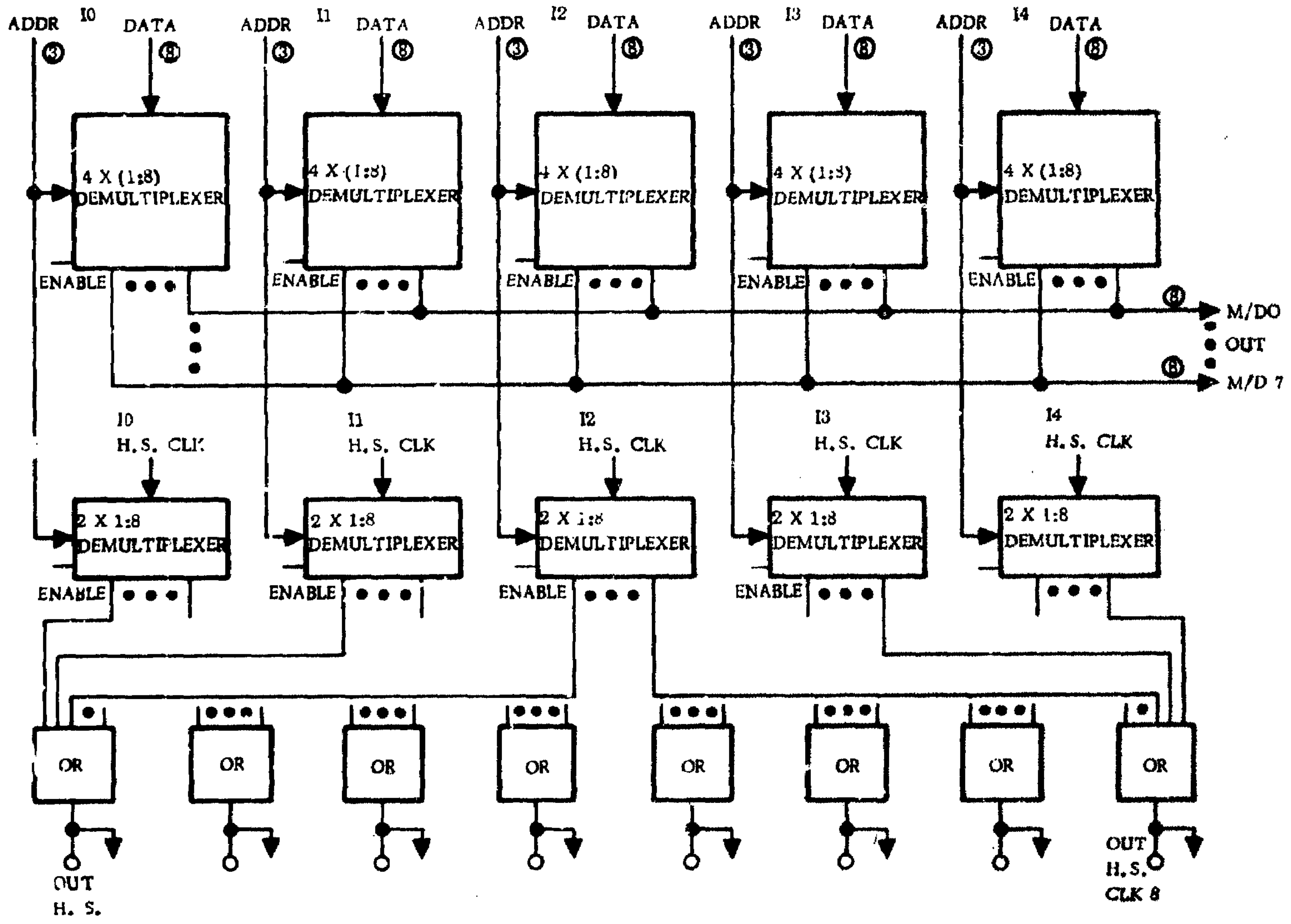


Figure 3-10. 5-Channel Output Switch Network Address Block Diagram

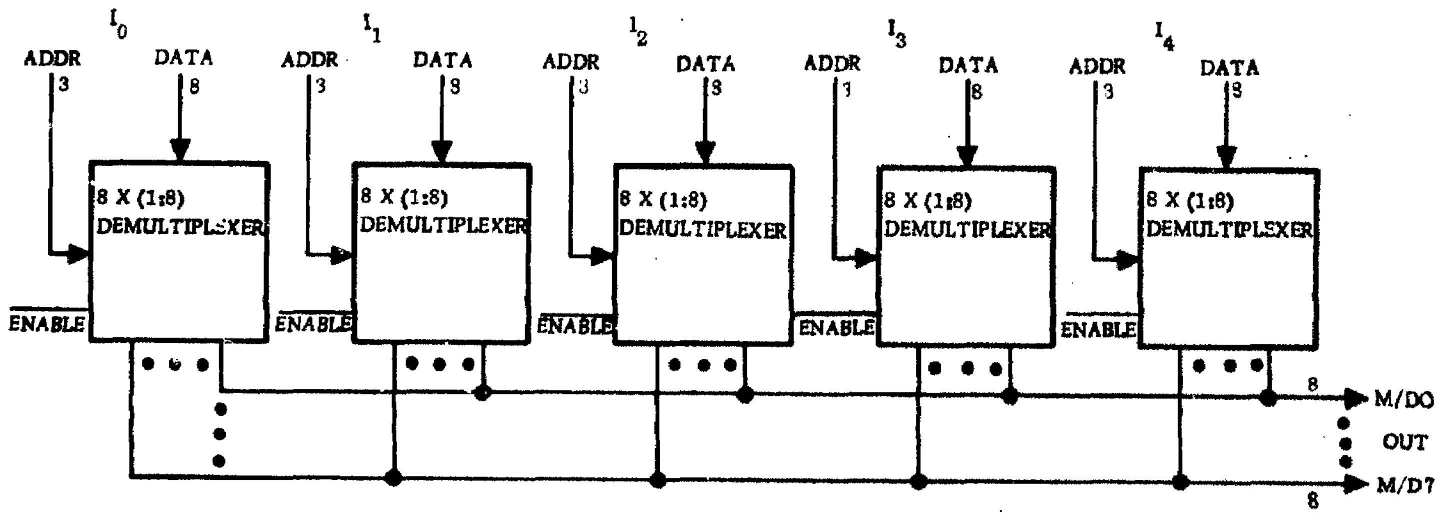


Figure 3-11. 5-Channel Output Switch Network (Data) Block Diagram

3.3.2.1 Memory Operation

The first type of signal from the MDC is a "memory operation request" signal to the MC. This initiates the comparison and priority logic in the MC. When the MC has granted access by that interpreter to the memory module it was requesting, a compare signal is returned from the MC to the MDC. This will send a clear pulse to the memory interface logic through the memory OSN and will initiate the setting of SAI and the transmission of high speed clocks to the output shift registers of the interpreter and through the OSN's to the memory interface.

In the case of a memory write, the input/output counter in the MDC will count four output high speed clocks and will then stop them.

In the case of a memory read, output high speed clocks are not counted. Instead, these high speed clocks are continually sent to the memory module interface. This interface will count four clocks coming in to it and will then initiate a memory read. Upon return of a completion signal from the memory, the memory interface will load its output shift registers and then allow four of the high speed clocks that are still coming through the OSN to clock these output shift registers and to be returned to the MDC and the interpreter with the shifted out data. The MDC counts four of these memory return clocks and will then stop the high speed output clocks and set RDC indicating that the data has been shifted into the interpreter input shift registers and is ready to be strobed into the B register.

3.3.2.2 Device Lock and Unlock

The second type of signal emanating from the MDC is a device lock or device unlock request sent to the DC. After the DC has accomplished this, a signal is returned to the MDC in order to set SAI and the operation is complete.

3.3.2.3 Device Read and Write

The third type of signal from the MDC occurs for device reads or writes and is sent to the DC to check the lock status of the device being addressed by the BR1/BR2 of the interpreter before proceeding. After it is confirmed that the device is locked, the DC returns a locked signal to the MDC. This will have the same effect as when a memory module is obtained, i.e., a clear pulse is sent to the device interface logic through the device OSN and initiates the setting of SAI and transmission of high speed clocks to the output shift register of the interpreter and through the OSN's to the device interface.

However, the distinction made between memory reads and memory writes is not made for devices. Both cases act like a memory read; i.e., for a device write the MDC does not stop the outgoing high speed clock after four clocks and indeed does not even count them. In both cases the device interface counts four clocks coming in to it and then stops accepting high speed clocks. In the case of a read, the device interface waits for some kind of "data available" signal from the device which it will use to load its output shift registers and to allow four high speed clocks which are still arriving from the OSN to clock these output shift registers and to be returned to the MDC and the interpreter with the data. The MDC, as for memory reads, counts return clocks and will set RDC.

In the case of a write, the response is very dependent upon the particular device being interfaced. In the case of a card reader, Burroughs sent back the next four high speed clocks to the Interpreter. In the case of a printer, Burroughs used a signal saying the last character was accepted by the printer to cause the device interface to allow return clocks. The four return clocks are counted by the MDC and used as a means of saying that the device accepted the data sent out.

4. MODULE ANALYSIS

This section summarizes the investigation of the Burroughs Multiprocessor architecture and its modules, in particular the interpreter and SWI modules. The machine's capabilities, limitations, and some recommended improvements are given in this section.

4.1 INTERMEDIATE (S) LANGUAGE CONSIDERATIONS

4.1.1 Definition of Approaches

Interpreters are microprogrammable. They have no order set and no specific data structures. They are specialized by replaceable microprograms for the various roles they must perform. Firmware is the word used for microprograms that will reside within a control memory of a computer. Firmware specializes the logic design for a specific purpose.

An "S" language of an Interpreter is equivalent to the object code or assembly language of a conventional machine. Each "S" instruction is equivalent to a machine instruction in a conventional computer. An "S" instruction may be as simple or as complex as the system requires (e.g., A NOP may be an instruction; so may an entire matrix multiply program).

The interpreter based systems execute their programs under control of microinstructions. These microinstructions are derived from the languages used by programmers to program the required tasks. Intermediate languages (called secondary or S languages) are utilized in the main memory to provide the source for the microinstructions.

The selection of an S language for the ASB avionics system for processing in the Burroughs Multiprocessor can be the subject of a lengthy study in itself. The answer is probably only achievable through iterative analysis involving language definition, trial programming, and evaluation of resultant statistics.

The types of S language to be considered for an ASB avionics system can take on many forms. Three possible S languages or modes of operation which appear reasonable to consider are:

1. Emulation of existing or hypothetical machines

2. Direct execution of high level languages

3. Microprogram optimization of the machine to the problem or application

These three modes of operation are described as:

Emulation - Programs written in some other machine(s) language can be executed by an interpreter. In this mode, the microprogram memories contain or are provided with the microinstructions for each of the emulated machine instructions. A fetch micro-routine is used to acquire these for decoding and transferral of control to the correct microinstructions. Operands are fetched and routed to either the available

PRECEDING PAGE BLANK-NOT FILMED

hardware working registers or main memory locations assigned to be equivalent to those of the emulated machine. Emulation may also be used to evaluate new or proposed machine designs, thereby allowing such designs to be tested/evaluated before being actually built.

Features in an interpreter which aid emulation capability are:

a. Modular word length
b. Fast shift network
c. Flexible microinstructions
d. Zoned bit selection
e. Ease of communication with external devices and memories
f. Multiple interpreter usage via the SWI

The emulation mode of operation was investigated in detail by emulating the IBM 4II Avionics Computer and determining the Burroughs Multiprocessor throughput capability while operating in such a mode.

Higher level language (HLL) processing - The program in the operating memory consists of higher order language instructions. There exist many possibilities here with regards to the actual HLL used and the amount of preprocessing done on the language before placing the program in the multiprocessor for execution. Two extremes in the amount ot preprocessing are:

1. Total compiling leading to the generation of an S language similar to the machine language in a conventional computer.

2. Little or no preprocessing with an S language that closely resembles actual HOL.

In between these extremes lie many possibilities with many tradeoffs involved such as speed or efficiency of execution vs amount of main memory and microprogram memory required.

Optimized instruction repertoire - As with the emulation mode of operation, instructions are fetched, decoded, and micro-routines used to perform processing functions. The difference is however, that macros can be defined more suited for the particular task to be accomplished thus achieving a more optimum memory and speed match. In generating an optimum or problem oriented S language, a higher order language processing approach or an emulation (modified by macros) approach to operation can be considered. The use of macros to optimize the emulated IBM 4II computer was investigated and will be given later in this section.

4.1.2 Higher Level Language Processing

4.1.2.1 Language Selection

The selection of a particular HLL to use involves many factors. A recent study (Ref 6) conducted by the B-1 Division of Rockwell International examined the feasibility of using a common higher-order programming language for the total computer complex and related avionics functions; and, if practical, attempted to determine the language best suited for this application.

As a result of this study, it appeared feasible to apply a common language to the areas of flight programming, mission software, and special support software. For hardware testing, requirements were so different, few of the common language candidates have the capability of handling the programs efficiently, and it is believed a separate language may be preferable.

The study was carried out in the following steps: (1) Survey of available languages and their compilers; (2) Comparison of language characteristics and attributes from the standpoint of B-1 requirements; (3) Choice of several candidate languages for more intensive study, including the writing of test programs; (4) Compiling and executing the test programs on equipment comparable to the planned B-1 computer system; (5) Analysis of test results in view of selecting a language to be used for all B-1 software.

The recommendation as a result of this study was to use a modified version of JOVIAL with SPL MARK IV as the alternate choice. Five paragraphs are quoted from Ref 6 which summarize the results from the study:

> "Several features of JOVIAL make it attractive for this prupose; namely, the concept of the COMPOOL for data management, the use of Tables for related data of different types, the ability to pack data in a computer word and address it directly, the provision for fixed point arithmetic if required, and the ability to manipulate bits and bytes. There are deficiencies in the language as it exists today; however, the USAF sponsored JOVIAL Committee is in the process of evaluating and revising AFM100-24, Standard Computer Programming Language for Air Force Command and Control Systems, dated 15 June 1967. Rockwell International is participating in this effort as an observer. It is anticipated that the "new" JOVIAL will adequately fulfill the tasks required by the B-1 system software."

> "PL/1 contains many desirable features for use as the B-1 Common Language; however, it is felt that the advantages of using PL/1 were not great enough to justify its selection over JOVIAL or SPL. The reasons for the decision was its incompatibility with the existing COMPOOL structure used extensively by SAC, the lack of available compilers other than the IBM-360 series computer, the high cost of compiler development for new computer equipment, and the compiler inefficiency as shown by B-1 benchmark problems executed on the IBM 360 series."

> "SPL appears to have a good future in this area. The SPL MARK IV version appears to best suit the needs of a B-1 Common Language, but due to its very recent development, little is known about its compiler efficiency."

> "CMS-2 exhibits many of the desired characteristics but appears to have no great advantage of JOVIAL, and the additional training of programmers and the development of new compilers does not seem to warrant its use for the B-1 application."

"The U.S. Air Force JOVIAL Standards Committee is actively working towards an update of JOVIAL to remedy the deficiencies currently recognized in the language. The results of this effort will be reflected in a major revision to AFM100-24. Thus, the recommendation at this time is the designation of the "new" JOVIAL as the B-1 Common Computer Language and SPL MARK IV as the alternate choice."

4.1.2.2 Preprocessing to an S Language

The justification for the use of some amount of preprocessing on the selected higher level language is that of speed improvement possible over the straight forward use of the HLL. As noted in Ref 7, without preprocessing, program statements would have to be scanned forward and in reverse in order to interpret their meaining at the time of execution. Preprocessing also permits the conversion of statements to codes more readily or efficiently processed and assigns addresses to variables and constants. Further, redundant statements or characters can be eliminated.

Therefore, preprocessing tasks can be categorized as follows:

1. Editing - This streamlines the program stream to ease real time processing. This is accomplished by removal of comments or blanks from source programs; conversion of expressions to more convenient forms (as reverse Polish) for processing; error checking; and program optimization.

2. Tabularizing - The tabularizing task places information into readily interpreted forms. This includes the identification of source language elements, insertion of pointers in compound statements, replacing constants with internal forms, and allocation data to static or dynamic environment.

3. Encoding - The generation of compact internal codes is accomplished with this preprocessing task. This requires the translation of operators, delimiters, and keywords to code and expansion of higher level features into lower ones if advantageous to execution results.

4. Address Forming - This task provides the conversion of labels, procedural calls, and variables with addresses.

It is noted that these functions can be split between preprocessing or real time processing in a number of ways. The allocation is dependent upon the memory and execution time factors and the amount of abstractionintroduced in the resultant S language and its affects on the HLL use advantages including traceability.

Some of the factors that have to be considered in defining a preprocessing approach are the method of structuring and processing the data. Much of the recent advanced work in this area has centered on using stack mechanisms, reverse Polish notation and descriptors (Ref 7, 8, and 9).

A stack is a storage mechanism in which the contents are accessed on a last in, first out basis. It has been used in the design of computers by Burroughs (B5500, B6500, B1700), DEC (PDP10, PDP11), and XDS (Sigma 7).

The dynamic behavior of stacks has been stated (Ref 8) to be well suited to the mechanization of recursive procedures and aids in:

1. The management of nested subroutines or procedures.
2. The efficient execution of arithmetic statements
3. The compilation and/or execution of higher order languages
4. The dynamic allocation of memory space
5. The protection of program data.

Descriptors provide a means by which items such as variables, procedures and control words can be defined as to type, attributes, size, location, initialization information, etc. Their usage enables data identification, validity checking between operators and operands, location pointing and indexing, and the monitoring of status, location and condition of information. The automatic identification of type and characteristics of data at execution time results in retention of straight forward representation of information to that point and is stated to result in memory savings and speed enhancement (Ref 9).

4.1.2.3 Example of Higher Level Language Processing

Reference 7 presents the results of a study, to design a computer capable of processing an intermediate form (at a high level) of SPL/Mark IV. The objective of this study was to design a special purpose computer architecture that could efficiently process SPL at a high level and compare such a machine with a conventional computer. A brief summary of the approach to the "S Language" taken in the referenced study will be given here since it is felt some of these results may be applicable to the consideration of higher level language processing on the Burroughs Multiprocessor, particularly for an ASB avionics system since SPL is one of the two recommended HLL for the B-1 Systems.

The encoding task of the preprocessing in Ref 7 converted operators and other symbols into "tokens". A token was defined as a short binary string on the order of six bits representing SPL symbols. Table 4-1 presents the tokens defined to do the following:

1. To encode primitives such as operators ('+'), delimiters (') '), or keywords ('IF') which appear in the object (preprocessor output) program.
2. To simplify the architecture. For instance, due to packing, the Noops are needed to fill memory words following branching tokens.

Table 4-1. SPL Tokens Defined in Reference 7

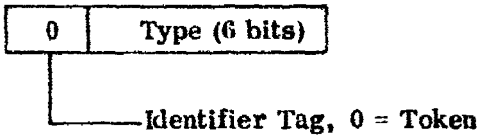
No.	Token	Meaning
1	ABS	Absolute Value
1A	F	Floating i.d.
2	AND	Bopr, and
2A	RF	Floating i.d. with rf
3	BIT	References bits of a variable
3A	FR	Floating I.D. rounded on assignment
4	BY	Loop variable increment
4A	RFR	Floating i.d. rounded with rf
5	BYTE	References bytes of a variable
5A	FD	Double precision (dp) floating i.d.
6	DECLARE	Begins a declaration block in a rr pred
7	ELSE	Begins the ELSE clause of a conditional stm
7A	RFD	dp floating i.d. with rf
8	END	Terminate IF, FOR, and LOOP UNTIL compound stms
8A	I	Integer i.d.
9	END DATA	Terminates a declaration block in a rr pred
10	EQ	Rel opr, equals
10A	RI	Integer i.d. with rf
11	EQUIV	B opr, equivalence
11A	L	Logical i.d.
12	FOR	Begins a loop stm
12A	RL	Logical i.d. with rf
13	GO TO	Direct GO TO stm
13A	T	Tentual i.d.
14	GQ	Rel opr, greater than or equal
14A	RT	Textual i.d. with rf
15	GR	Rel opr, greater than
15A	B	Boolean i.d.
16	IF	Complex IF stm
16A	RB	Boolean i.d. with rf
17	IND	Indicates following stm label is indirectly referenced as pred argument
17A	ARRAY	Array i.d.
18	LAND	Logical opr, product
18A	RARRAY	Array i.d. with rf
19	LOOP UNTIL	Begins a loop stm
20	LOR	Logical opr, sum
21	LQ	Rel opr, less than or equal

LEGEND:	rr	recursive or reentrant	B	Boolean
	i.d.	item declaration	Rel	Relational
	rf	repetition factor	A	Arithmetic
	opr	operator	pred	procedure
	stm	statement		

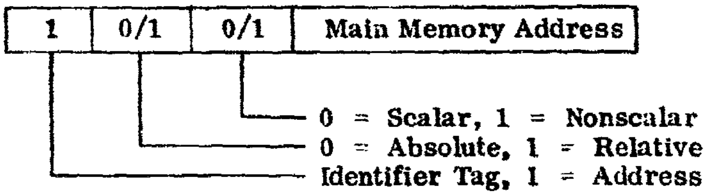
Table 4-1. (Cont)

No.	Token	Meaning
22	LS	Rel opr, less than
23	LSH	Left shift opr
24	~~LXOR~~	~~Logical opr, exclusive or~~
25	NOT	B opr, negation
26	NOOP	No operation
27	NQ	Rel opr, not equal
28	OR	B opr, or
29	ORIF	Begins subordinate conditional stm
30	RECURSIVE	Indicates a rr prcd
31	REM	Remainder of a division
32	RETURN	Prcd return
33	SGOTO	Switched GO TO stm
34	SIF	Simple IF stm
35	STOP	Computer halt
36	TPOSE	Matrix opr, transpose
37	UNTIL	Terminating condition in loop stm
38	o	Indicate prcd call
39	(	Delimiter
40	)	Delimiter
41	+	A opr, add
42	-	A opr, subtract
43	x	A opr, multiply
44	/	A opr, divide
45	**	A opr, exponentiation
46	/*	Matrix opr, cross product
47	/*/	Matrix opr, multiply
48	,	Delimiter
49	=	Assignment opr.
50	==	Exchange opr

A format for these tokens was defined as

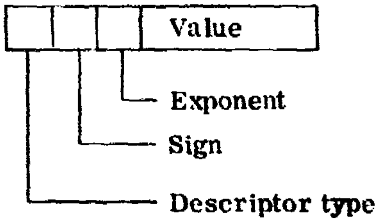
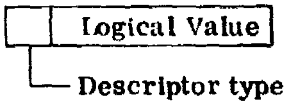
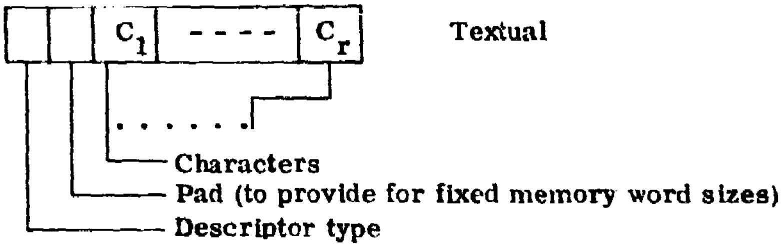


The tokens are used with address words to describe the program. The format for address representation to acquire item names, constants, procedures and statement names is


1
0/1
0/1
Main Memory Address
0 = Scalar, 1 = Nonscalar
0 = Absolute, 1 = Relative
Identifier Tag, 1 = Address

These preprocessed program elements are provided to and interpreted by the hardware to accomplish the program execution. This includes the unpacking of tokens and addresses and translation of these into a sequence of commands and data addresses.

It is to be noted that the data in the referenced study is stored using descriptors bits assigned to each data word to identify the word type. The following table defines the eight descriptor types and three classes of data to represent the program information.

Class	Format	Descriptor Type
1		Floating Point, unrounded Floating Point, rounded Floating Point, double precision Integer Boolean Nonscalar (arrays)
Class	Format	Descriptor Type
2		Logical
3	 C_1 - - - - C_r . . . . . . Characters Pad (to provide for fixed memory word sizes) Descriptor type	Textual

It is felt an S language as described above from ref 7 can be executed on the Burroughs Multiprocessor. With regards the particular level used for SPL, a significant amount of effort would be needed to determine if the level used in Ref 7 can be applied efficiently to an avionics processing application.

The execution of a preprocessed program represented by an S language as defined by the tokens in Table 3-3 on the Burroughs Multiprocessor would involve the fetching of the program element and the analysis of statements to obtain commands and sequences to be performed by the primitives of the interpreters.

A program fetch routine consisting of micro instructions would access the main memory for program words containing tokens or address elements in a sequential manner. Since eight bit bytes or words with memory packing would be used, the fetch routine must extract the individual elements in proper order. The routine must be initialized correctly by some external means. Further, the routine must respond to requests for transfer of control to alternate locations within the programs as required during execution.

The analysis of the sequence of addresses and tokens obtained as the result of the fetch routine would le ιd to the translation of these into executable commands. A concept of translation of program information to microcontrols is shown diagrammatically in Figure 4-1.

A token initially establishes a parsing state which determines the need for additional tokens or semantic routines. Commands are provided to control routines from either the state analysis or the semantic routines. These commands are translated to primitive interpreter operations using the data addresses as required and calling upon the fetch routine for additional program elements or to control program transfers.

Further, in-depth considerations with regards to HLL processing can be found in Ref 7. Flow charts, processing logic, and stack features were defined for conversion of preprocessed SPL into executable commands. It is felt some modifications to the referenced study, such as converting expressions to reverse Polish notation would be necessary for efficient S language execution.

4.2 INTERPRETER MODULE

4.2.1 Word Size

The word size to use in the Burroughs Multiprocessor depends upon many factors including whether emulation or higher level language modes of operation are to be used.

The Burroughs approach has been to design modularity based upon 8-bit bytes. It should also be noted that a 16-bit word length is desired for the microprogram memory word. One reason given for this is to maintain compatibility with popular military memory word sizes. Figure 4-2 presents two plots of data extracted from reference 10 to support this statement.

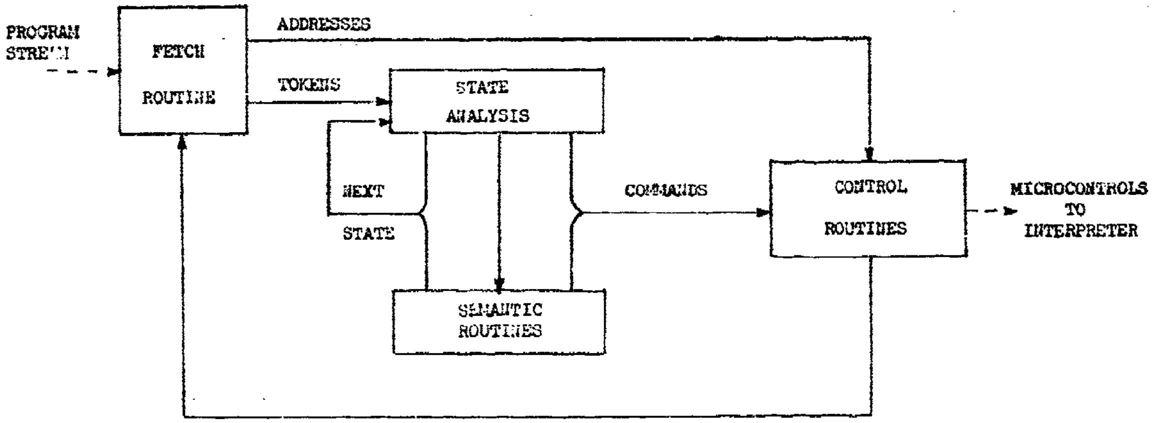


Figure 4-1. S Language Translation

Of the 52 machines presented in Ref 10 in deriving Figure 4-2, it is apparent that a majority (~75 percent) is of the 16 or 32 bit single word, 8/16/32 multiple word, or 24 bit word classes. This data is for machines as of mid 1970. Since that time, the trend toward these sizes is even more evident from considering the 16/32 bit computer designs for the F15 and B1 avionics.

Both the instruction format and data requirements affect the selection of a word length. The 16/32 bit data word has been found to be suitable for avionics applications. Instruction word length is a function of the number of instructions necessary to fully utilize the logic capabilities, the addressing requirements, and index register designators. It is usually desirable to make the data and instruction words the same length so that they can be stored interchangeably in memory. If higher precision is required in portions of an application, double word length operation is used.

It would be informative to examine the quantity of modern military computers produced to discover popular word sizes on that basis. These numbers are not available. Examination of Ref 10 and knowledge of the applicable military programs leads to the conclusion that the above word sizes would also represent a high percentage of the latest computers produced.

One advantage of higher level languages is the semantic conciseness possible. Information can easily be conveyed in byte oriented instructions, descriptors, and data streams. Compatibility between any HLL avionics system and ground based machines is desirable in order to simplify program and machine checkout and simulation. An eight bit byte is considered standard within the commercial computing industry.

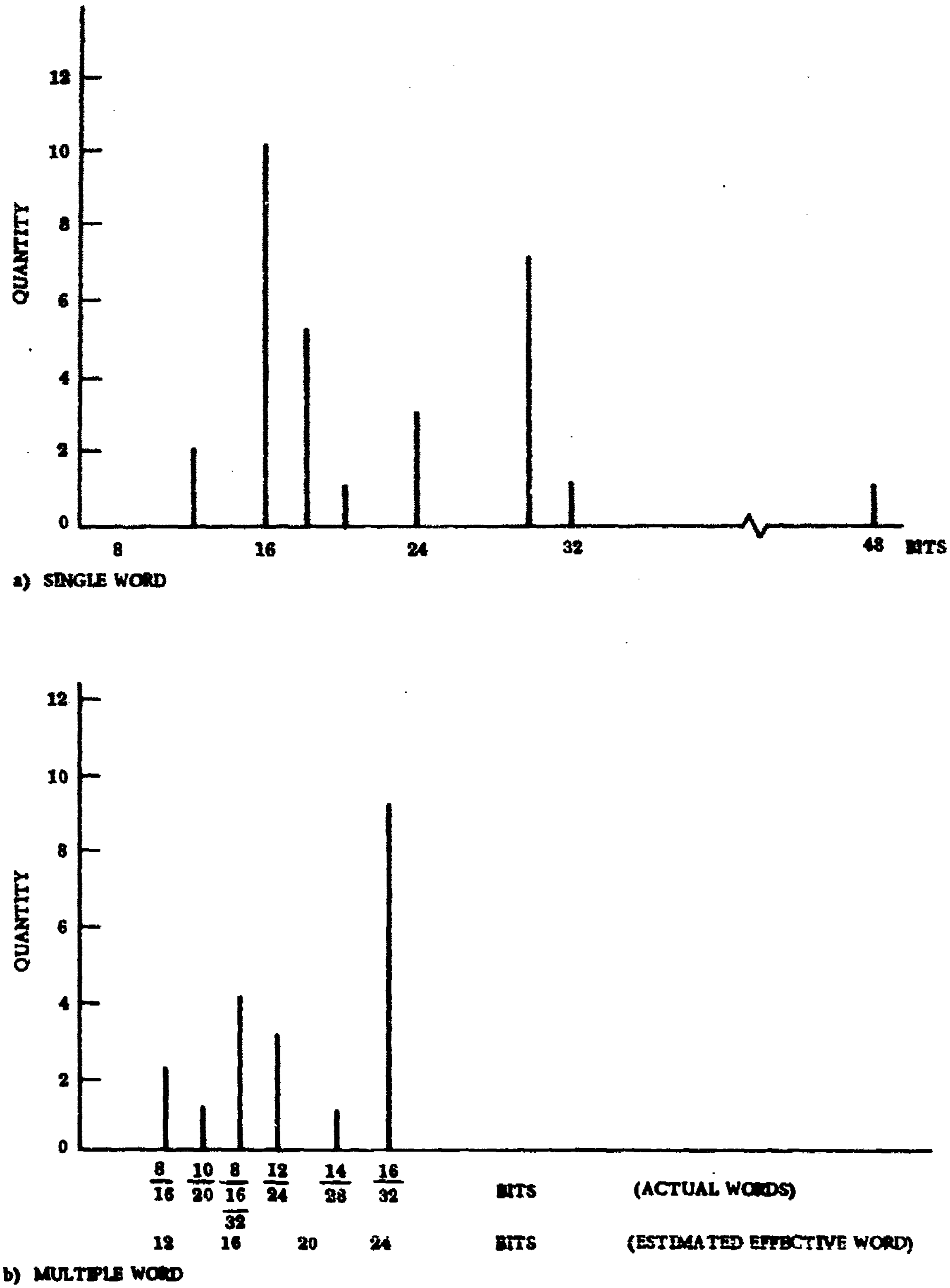


Figure 4-2. Military Computer Instruction Word Sizes as Reported in Ref 10

The conclusion derived from this examination is to utilize an 8-bit byte and multiples of it for Multiprocessors operating in a higher level language and 16, 24 and 32 bit (including combinations of 16 and 32) for Multiprocessors operating in emulation.

4.2.2 Microprogram Memory Organization

4.2.2.1 General

The source of microprogram instructions may be via single or multiple memory levels. The advantage of the latter has been stated by Burroughs (Ref 3) as requiring less total bits. The purpose of this discussion is to examine this area in order to establish the various factors involved.

This memory reduction is derived from a realization of common microinstructions within different S instructions or macros. The key factor is just how much replication is expected. Further, it is to be recognized that even with one memory level it is possible to utilize, through branching, replication of microinstructions. The overhead expense for this branching is a function of the order (sequential or random) of the microinstructions and the similarity of the macros.

The number of macros expected ranges from 20 to 60, with a likely number being 40.** Further, an estimate of the range of microinstructions per macro is 10 to 40 with the likely average value being 20.** The plot, Figure 4-3, shows the total bits required when implemented with a single (and no replication factor) or a dual memory approach for various replication factors with a microprogram memory sizing approximating the Burroughs design.

**Support for the estimates are from the following:

1. "Dynamic Microprogramming," A.B. Tucker and M. J. Flynn, CACM, April, Vol. 14, No. 4, pp 240-250.

   ("To adapt the microprogrammed processor to a particular need, an appropriate collection of macros is selected. Typically this might contain 40 macros.")

2. Phone conversation with Burroughs indicated 15 microinstructions per macro estimate.

3. Trial programming in an emulation mode (to be presented later in Section 4) showed a typical estimate of 40-45 microinstructions per macro.

4. "Microprogramming Environment on the Burroughs B1700," Wayne T. Wilmer 6th Annual IEEE Computer Society Conference, San Francisco, September 12-14, 1972.

The following Table 4-2 is extracted from this reference:

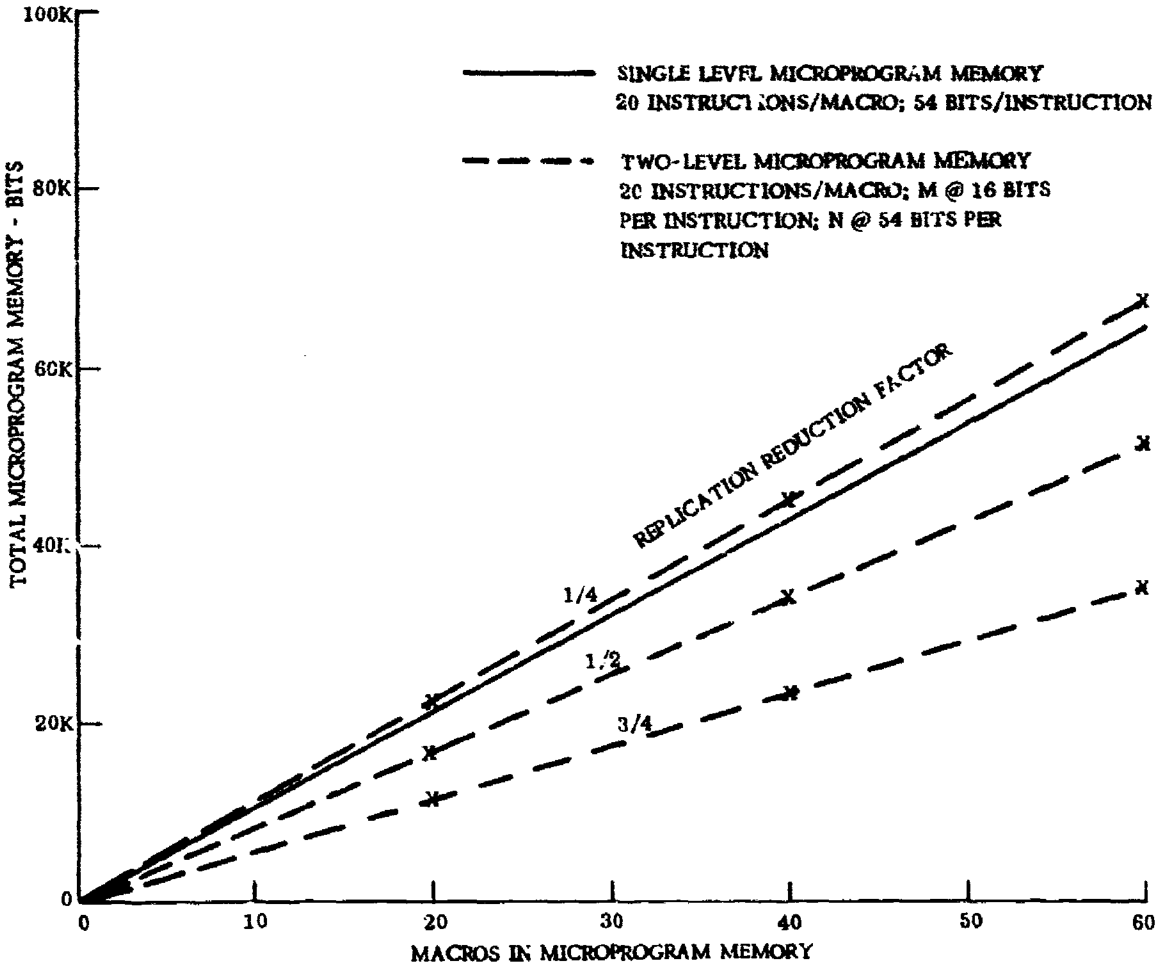


Figure 4-3. Relationship Between Amount of Replication Within Macros and Total Microprogramming Bits

Table 4-2. Microinstruction Requirements

Abstract Machine	Number of Virtual Instructions	Number of Micros per Instruction
Second Generation	25	67
FORTRAN	41	39
COBOL	42	39
Fourth Generation	74	25

The observations which can be made from the considerations in this section are:

1. The replication reduction factor must be greater than 1/4 in order to obtain a reduction in the total number of bits over a single level (straight line coding shown for single level—this may be reduced by branching between macros using commonality also, thereby increasing the replication factor required).

2. Questions with regard to how much replication and the order of micro-instructions depends upon the macros to be implemented for both single and multiple memory levels, and can be answered only after the macro's definition.

3. As Burroughs observed, the use of multiple level requires either faster memories (at higher costs) or a speed loss. The cost effectiveness is dependent upon the amount of memory reduction and cost per bit.

4. The amount of logic is about the same with either approach.

4.2.2.2 Two Level Microprogram Word Size

Two levels of microprogram memory can save total memory bits if there is a repetitive demand for microinstructions (as discussed previously). Further, advantages can be obtained by a two level type if the longer single level word requires too much interface logic. The factors involved in the selection of an appropriate word size for each of the two levels are discussed in the following paragraphs.

4.2.2.2.1 Microprogram Memory (MPM) Level. The considerations involved in the selection of a word size for the MPM are as follows:

1. Satisfaction of the functions to be performed

   a. Addressing - With a nanomemory and MPM address capability of 4096 words, 12 bits are required.
   b. Register Loading - The shift count register (4, 5, 6 bits for 16, 32, 64 bit data words, respectively) and the literal register (8 bits) must be provided with information. Further, Burroughs indicates these registers are loaded together often enough to warrant both being in the same word.

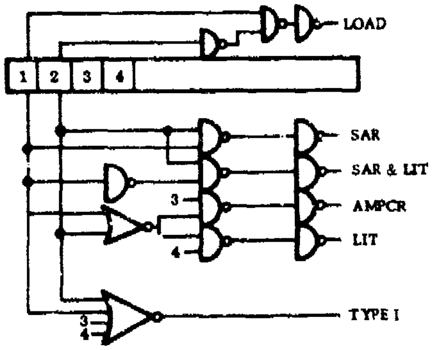
c. Word Type Identification - The addressing and register loading establishes the need to supply bits to identify the operation involved. Burroughs has classified the MPM words in two types. Type 1 provides the direct addressing of the nanomemory; Type 2, the register loading operations.

2. Compatibility with the S memory. Most popular computers have been organized with operating memories of either 16 or 32 bit word sizes. The result is that many peripheral devices and memory units are of these word sizes. Further, if byte processing is provided, 8 bits is preferred. Thus, the desirable MPM word size from the viewpoint of storage in the S memory is one of these. Since the 8 bit word size is too small to contain the information to be conveyed without excessive memory accesses and the 32 bit too large and leads to too many unused bits, the 16 bit word size appears to be preferred.

The organization of a 16 bit MPM word size revolves around the above requirements and the minimum number of gates to do the type decoding in order to supply the control signals. Flexibility to accommodate additional instruction types and levels of logic or delays are other considerations to be made.

Without considering the flexibility needs, a slight improvement in the Burroughs MPM word format may be obtained by modifying it as next discussed.

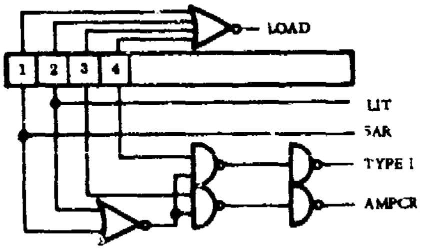
Decoding the MPM format requires the following or equivalent logic:



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

10	LOAD MPM	
11	SAR	
01	SAR	LIT
001 -	AMPCR	
00-1	----	LIT
0000	NANOADDRESS	

THE MODIFIED FORMAT REQUIRES:



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

0000	LOAD MPM	
1 -	SAR	-
- 1	-	LIT
001	AMPCR	
00-1	NANOADDRESS	

It is obvious that the latter format simplifies the decoding.

The conclusion reached is that the 16 bit word size for the MPM is suitable.

4.2.2.2.2 N-Memory. The N-Memory word size is dictated by the number of control and condition signals required and the extent to which coding of the bits to indicate these are used.

A longer word is required when minimal encoding of bits is used. This is obviously a disadvantage with a large number of control points. Table 4-3 shows the Burroughs' word and an estimate of the control points.

The Burroughs organization of the N-word reflects considerable design thought. The capability to provide some operations in parallel is included. However, it it noted only one condition can be tested at a time. To be able to do more would required more additional bits in the Nanomemory word. The value oɪ limitation of these features requires examination by trial microprogramming.

The possibility of reducing the number of decode gates by increasing the N-word size was examined. Table 4-3 presents information showing the total number of control or conditions as being 139. This word size would be required if no decoding were used. Obviously encoding is desired to reduce the word size. Table 4-3 contains an estimate of the number of Series 5400 integrated circuits required to decode each field of the nanoinstruction. An alternate word is shown which uses less encoding and more bits while retaining the same parallelism. This approach indicates an increase of 16 bits in the N-word could lead to approximately 30 less integrated circuit components. With a density of 256X1 (as used by Burroughs) the component count between the 54 and 70 bit words remains about the same for a 512 word Nanomemory. However, present day densities of 2 to 8 K bits at the required speeds are available and would enhance the longer word usage.

The above ar.alysis leads to a recommendation to consider expanding the N-word size. A more extensive study which includes the consideration of interconnections, partitioning, currently available integrated circuits, and power should be made.

4.2.2.3 Single Level Word Size

The considerations for the word size for a single level microprogramming capability are similar to those for the two level. These are:

1. Addressing - Providing addressing capability for 4096 words defines the need for 12 bits in the word.

2. Register Loading - The shift count and literal registers require 6 (for 64 bit data) and 8 bits respectively.

3. Word Type Identification - With a longer word, less types and less identification bits are required since more than one or two information fields can be included in a word type.

Table 4-3. Microprogram N-word Assessment

				Alternate Longer Word	
N-Bits	Function	Bits Used	Condition or Controls	Est. Difference in IC's	Bits
1-4	Condition Tested	4	16	-	4
5	Condition Value	1	2	-	1
6	Logic Unit Conditional	1	2	-	1
7	External Conditional	1	2	-	1
8-10	Condition Adjust	3	8	3	7
11-13	Successor Instruction for Condition	3	8	-	3
14-16	Successor Instruction for not Condition	3	8	-	3
17-19	Adder X Input	3	8	3	7
20-26	Adder Y Input	7	18	2	9
27	Inhibit Carries into Bytes	1	2	-	1
28-31	Adder Operation	4	16	8	7
32-33	Shift Type Selection	2	4	1	3
34-36	A Register Input from BSW	3	4	1	3
37-40	B Register Input Select	4	10	5	6
41	MIR Input from BSW	1	2	-	1
42	AMPCR Input from BSW	1	2	-	1
43-45(46)	Memory/Device Address Input	4	7	3	3
47-48	Counter Input	2	4	1	2
49-50	SAR Input	2	3	1	2
51-54	Memory/Device Operation	4	13	2	5
Totals		54	139	30	70

4. Compatibility with the S Memory - The word sizes possible for the single level microprogram memory according to this factor are 16, 32 and 64 bits. The latter is felt to be too large and could lead to many unused bits for the register loading type instructions. Further this size could introduce more logic and/or complicate the loading (for read/write MPM) and addressing of the microprogram memory. The 16 bit word size would require multiple microprogram memory accesses and may be worth investigating if, upon examining the timing and the 32 bit size, it appears feasible.

The single level microprogram word size of 32 bits can have a number of different formats. The following is not meant to present the best but to define a workable set.

Format A	Type 2	A M P C 12	EXT 2	NEXT ADDRESS 12	EXT 2	SPARE 2	Type II

Format B	Type 2	NEXT ADDRESS 12	EXT 2	LITERAL 8	S A R 6	SPARE 2	Type II

Format C	Type 1	1st half of microcontrols - 30	Type I
Format D	Type 1	2nd half of microcontrols - 30	

The timing with a single level memory can nearly approximate that defined for the two level. The microprogram memory access required for a 32 bit wide single level approach can be accomplished as with the two level. This means two accesses for a Type I and one for a Type II microinstruction.

Phasing details can follow also. As an alternate, it could be possible to split the two accesses for the Type I, and do one during each of the Phase 1 and 3 timing. This approach may offer a slight improvement in speed especially if there are a lot of unconditional situations.

4.2.2.3 Comparison Between One and Two Level Microprogram Memory

The one level, 32 bit word approach may lead to faster information transfer from the operating memory (for read/write MPM), can simplify the decoding of instruction types, and present a slight improvement in timing. Because both approaches would use microinstruction overlap, buffers are required. Additional logic would be required with the one level to accommodate the wider bus interface to the operating memory and the steering of the Type C and D format to the appropriate buffers. The net result is that both approaches have about equal amounts of logic. The one level approach can be made about 20 percent faster by:

1. Eliminating one memory access with the use of a wider (64 bit) microprogram memory word; or,

2. Accessing the Phase 3 control word during the logic conditional testing done in Phase 1.

As noted earlier, additional logic in the timing and selection of the interface to the operating memory is needed.

The conclusion at this stage of the study is that the Burroughs two level microprogram approach is a reasonable and cost effective approach. Change to any other depends on whether additional speed is needed.

4.2.3 Limitations and Possible Improvements

4.2.3.1 General

The most apparent limitation of the interpreter observed in this study is the throughput or speed capability. The throughput capability for an emulation mode of operation was calculated by trial microprogramming and will be presented later in this section. It was found that the interpreter had limited throughput capability in the emulation mode. There are two principal ways to improve the throughput capability. One is to use the machine with an S language that is at a relatively high level and containing a high degree of macros or complex operators. This reduces the amount of main memory accesses and also the amount of instruction fetch and decode overhead. The other method is to change the interpreter design, incorporating features that would enhance its throughput. The purpose of this section is to identify some of the possible changes in the interpreter design that would enhance its throughput.

4.2.3.2 Provide Temporary Storage Via MPM.

At present the MPM cannot be used to read out data for the interpreter. If this were possible, the MPM could be used to provide either additional interpreter registers or as temporary storage for data. These features could be particularly useful in processing a complex HLL. The modifications needed to the interpreter to provide this capability were briefly investigated.

A data input path to the MPM is needed, if it can only come from one place such as the BSW then a nanomemory select bit is not needed for this function. An address register is needed that would be used as an alternate when writing into MPM, no extra nanobits are needed since its use would be implicit. However the capability to load and operate on this address register would be required. If this register is brought in as a Z input to the adder then an extra nanobit is needed. Also if this register is loaded from the BSW then an extra nanobit may be needed depending on whether some of the spare codes in nanobits 43-46 can be used for this function. Finally an extra nanobit may be needed to specify the MPM write (possibly some of the codes in nanobits 51-54 could be used).

It should be noted that multiple write cycle capability will be required since the MPM is 16 bits wide and the interpreter is n x 8 bits wide (32 bits in this study). Therefore more than 1 MPM write cycle would be required for executing a type 1 microinstruction that specifies a MPM write.

In summary this function will require an address register, changes to the control section to allow MPM write cycles and delayed fetch cycles (in case multiple write cycles are required), and probably 2 extra nanobits.

4.2.3.3 Provide More Registers

At present there are three A registers and three nanobits that control their loading from the BSW (this allows them all to be loaded simultaneously). It is possible to provide seven A registers and only require one additional nanobit (to select the adder A input) if the ability to load all the A registers in parallel is given up.

More than three A registers would be a significant improvement in the interpreter. For example, from the experience in the emulation mode trial microprogramming this would:

1. Prevent having to store the emulated machines index registers in main memory.
2. Allow the emulated machines Q register to be kept in the interpreter
3. Allow these registers to be used for temporary storage in manipulating the instruction format.

Providing more A registers would obviously also enhance the performance in a HLL processing mode with the use of complex macros.

4.2.3.4 Program Counter

In most application a program counter mechanization is needed. The Burroughs multiprocessor requires one of the A registers to be used as the program counter. This is not very difficult to do since the logic unit is very flexible. However, since this function will normally be required, it may be more convenient to provide a separate counter. This could be provided as MR3 or MW3 in nanobits 43-46. Two additional nanobits would be required to control the counter:, no change, increment +1, input from BSW. If more A registers are provided than there would be less of a need to provide a separate program counter register.

4.2.3.5 Changes to Logic Unit

There are a number of items in the logic unit that will normally be required in an emulation mode that are not presently implemented and therefore difficult to perform in the present logic unit. One of these is the carry. At present this is a dynamic condition and is only recorded if tested and a condition bit set. It appears it would be more desirable to have the carry simply latch a FF if it occurs.

Another item is the shift control. It is frequently required to shift and spread sign, this is presently difficult to perform and should be added to the shift control. Also it is desirable to be able to record if a one was shifted out of the shift register for a shift of η places, no such capability exists at present.

The above items would result in more condition bits. It is also felt that more than three EX condition bits should be provided. More flexibility should also be provided in testing the condition bits. For example, a useful set of combinations would be test EX1 + EX2 +EX3, EX1 + EX2, EX1 + EX3, EX2 + EX3 etc. This would facilitate the mechanization and testing of interrupts. This would alleviate having to 'or' all interrupts in to one bit or having to proceed through a number of separate microinstructions to test a number of interrupts.

4.2.3.6 Buffer Storage

As an extreme to increase speed one might consider adding a buffer memory to operate as a cache. The actual effectiveness of a buffer would depend on the particular characteristics of the program being processed.

Features of buffers which must be considered in their design includes the manner of buffer control. Either direct mapping or associative methods can be used. (Ref 11). In the direct mapping, the S memory would be divided into blocks of information. Each of these blocks would have a tag and be assigned to a fixed block location in the buffer. In a fully associative mapping, any block in the S memory can be loaded anywhere into the buffer. The blocks are tagged and everyone is searched to determine whether an addressed block is in the buffer. This method gives more flexibility at the cost of time and search logic.

Other, less associative methods as described in the reference can provide performance with reduced flexibility, but with less hardware cost than the full approach.

Another consideration in the use of buffers is that of what is the best manner of writing data into the buffer in order to eventually update the S memory. Two ways of doing this are storing through and periodic block update. The storing through method requires an S memory write for every buffer one. The block update method permits the accumulation of the updated data in the buffer and a write into the S memory only upon replacement of the block which has had data written into it while in the buffer.

Other aspects of the buffer to be examined in a detail design include the buffer replacement algorithm. It should be simple, such as based upon activity. Fetch anticipation is felt not to be necessary.

Studies to date (Ref 12 and 13) indicate the best typical sizes for the buffer memory to be 2 to 4 K words with blocks of data of 4 to 16 words.

A block diagram of a possible buffer is given in Figure 4-4. The functions of the major components are as follows:

1. Buffer storage control. This logic provides the associative search control to determine if the data requested is in the buffer and to provide pointers for addressing any stack or file structures.

2. Address array. This storage contains the address information of the data stored in the buffer.

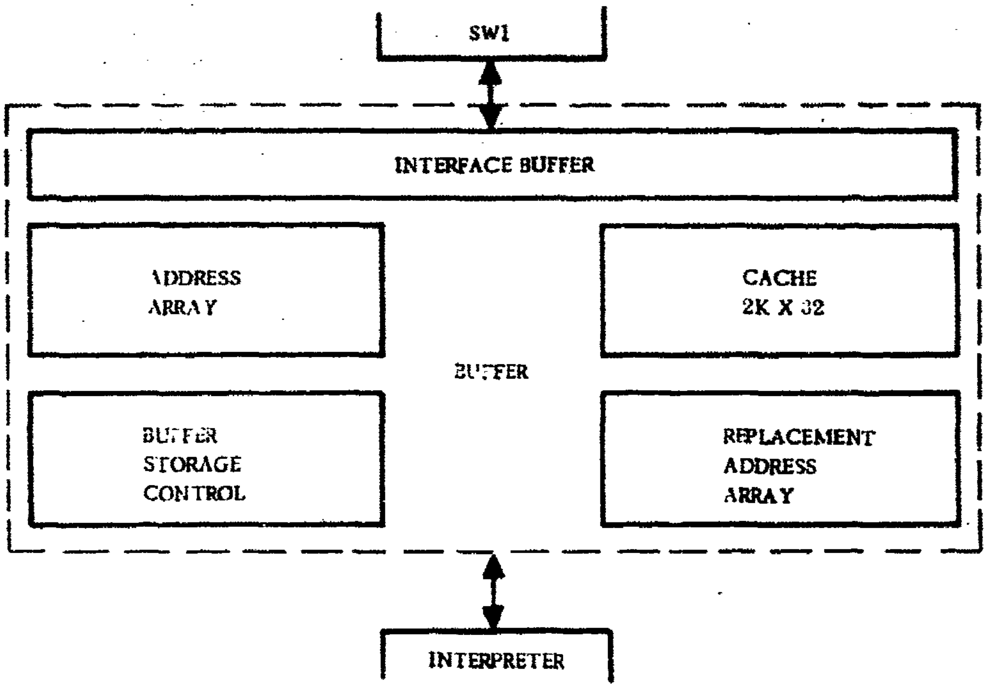


Figure 4-4. Buffer Storage

3. Replacement address array. This storage provides an activity source for determining the buffer replacement in the event of addressed data not being in the buffer.

4. Cache. This memory holds the instructions and data used in program execution. It is transparent to the programmer and maintained dynamically. This storage can also provide the temporary storage and stack locations.

5. S Buffer. This buffer accepts four to sixteen (4 x 8) word blocks from the S memory.

4.2.3.7 Multiply Capability

It became apparent in considering the interpreter in the emulation mode that the speed of multiply is a limiting factor in the present design particularly for an avionics application. The fastest multiply algorithm developed by Burroughs for use in the present design is given in Ref 1. It is noted to consist of the following:

		Clocks
1.	Sign determination and conversion to positive values	2
2.	Setup of counter and registers	3
3.	Multiply Loop, averaged for 32 bit data	80
4.	Sign conversion to sign magnitude	2
	Total	87

With a 4 MHz clock this is an average multiply time of 21.75 microseconds, to thı, must be added the instruction and operand fetch times and instruction format decode. This results in multiply times on the order of 30-35 microseconds.

The recommendation at this point in time would be to strongly consider faster multiply algorithms, such as 2 bit at a time or 4 bit at a time, in future designs of the interpreter.

4.2.3.8 Logic Speed

The present interpreter clock is 4 MHz. It is quite possible that a higher speed clock could be used after carefully analyzing future logic designs. This would have a direct affect on interpreter speed.

4.3 SWITCH INTERLOCK MODULE

4.3.1 SWI Timing

The logic diagrams of the SWI modules were analyzed to determine the time required to complete read/write operations for memories and devices. A summary of the results is contained in Table 4-4.

Detailed timing charts are contained in Figure 4-5 through 4-7 for memory and device operations. The timing charts are for the SWI only and essentially start from the time the nanobits are clocked into the MDC register. Any other time required to setup the nanobits between the interpreter and the SWI must be added to all memory/ device operations. Likewise, a specific memory/device cycle time has not been assumed and must be added to all memory/device read/write operations.

Figure 4-5 contains the timing charts for memory operations. The detailed sequence of events is first shown for a memory write assuming the new address (memory module) is identical to the old address. The changes to this sequence are then shown for read-old address, write-new address, and read-new address.

4.3.2 SWI Interface with Memories and Devices

The present design of the SWI requires certain interface logic to permit the transfer of data, address and control information to devices or memories.

Table 4-4. SWT Timing

	Time — Interpreter Clocks							
	SAME Address						NEW Address (ADDITIONAL)	
Operation	Nano Address Setup	SWI Setup	Address/Data Transfer Out	Data Transfer In	Mem/Dev Cycle	Total	Priority Resol.	Total
	*				**		***	
Memory Read		0.2	0.8	0.8		1.8 +	0.4	2.2 +
Memory Write		0.2	0.8	-		1.0 +	0.4	1.4 +
Device Rd/Wr		0.2	0.8	0.8		1.8 +	****	

* Dependent on interpreter design and clock waveform, maximum time would be 0.5 interpreter clocks from start of nanomemory cycle.

** Time for memory or device read/write cycle.

*** Time to resolve priority is dependent on the interpreter priority and if the memory is busy, the minimum time is shown.

**** Not applicable since a device must be locked to an interpreter before access is granted.

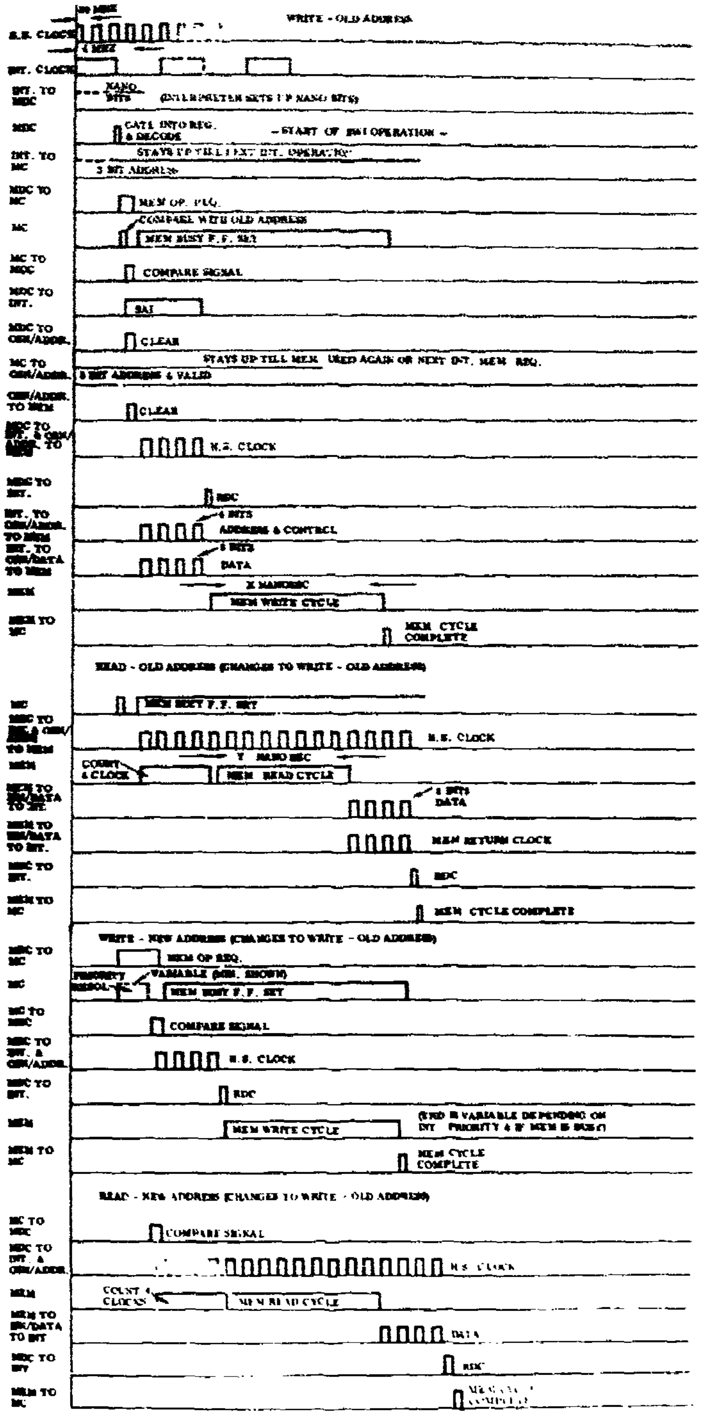


Figure 4-5. SWI Timing - Memory Operation

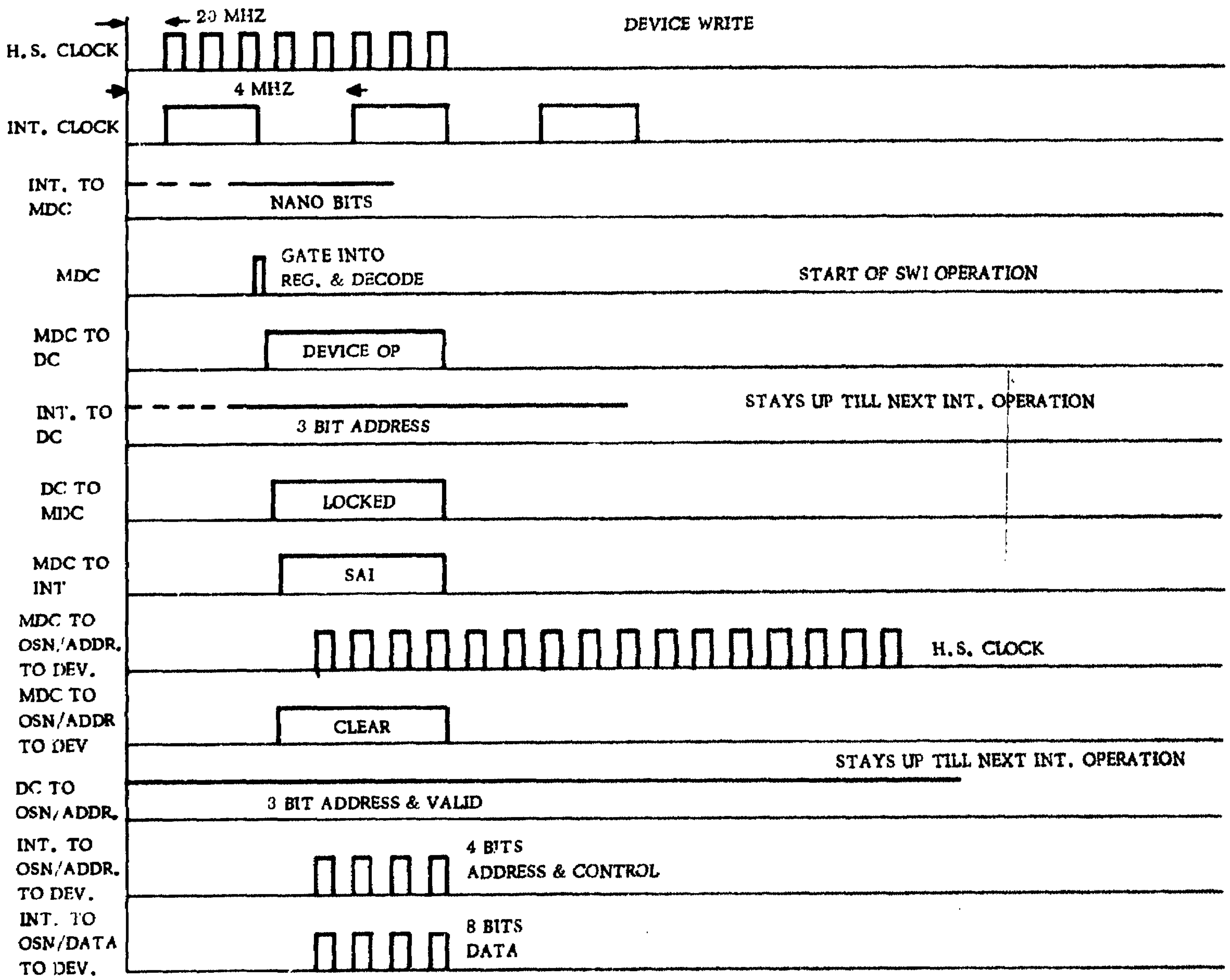


Figure 4-6. SWI Timing — Device Operation

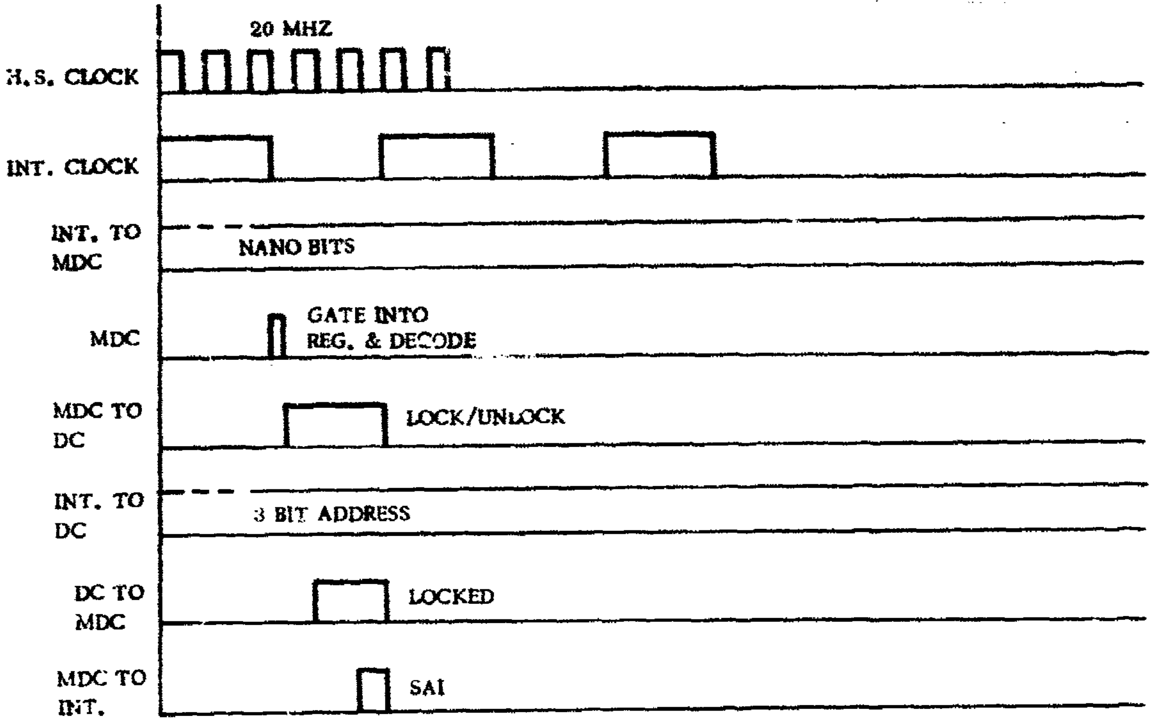


Figure 4-7. SWI Timing — Device Lock/Unlock

Figure 4-8 presents the Burroughs Multiprocessor concept for information transferred.

As shown the interface at devices and memories requires shift registers and a counter. Since the concept is to send the command (type of operation) in the address word, additional logic is required to extract this information.

The memory operations are initiated by the decoding of the nanobits and issuing a Mem. Opns. Req. from the MDC to the MC. If the address comparison priority and memory busy logic permits, the SAI/Clear signal is generated. The latter provides a reset for the counter and controls in the memory interface. Simultaneously, the high speed (HS) clock is enabled and transfer of the address and command information from the interpreter occurs. The clock pulses are counted and a memory initiate command generated. The read or write operation is determined by the decoding of the received address information.

If a write, the interpreter data is provided simultaneously with the address. There is no need to transmit any information back to the SWI. The number of bytes transferred is controlled by a counter in the MDC as shown.

In a read, the return HS clock (MRC) is returned after the memory has performed the command, counted in the MDC, and used to terminate the operation.

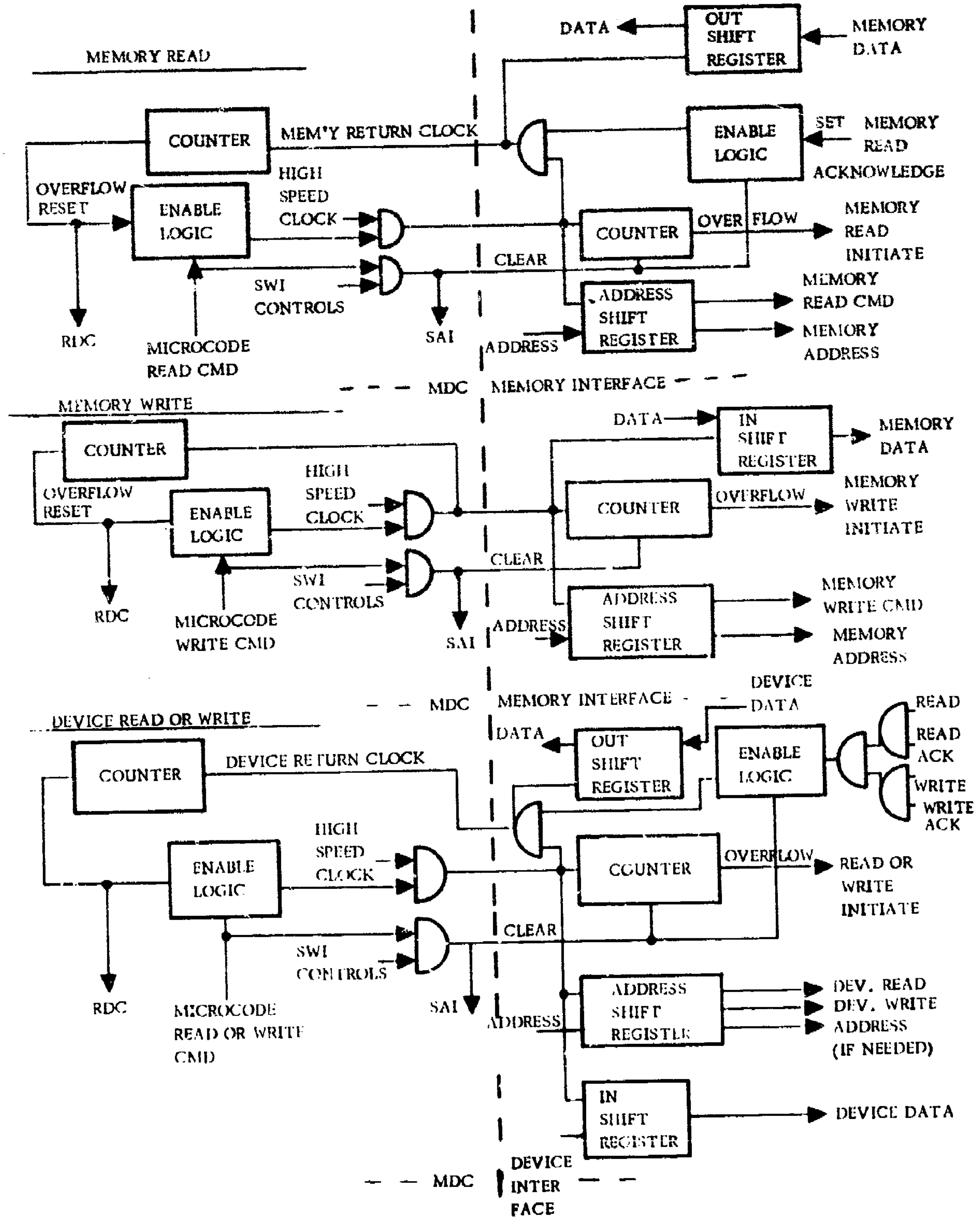


Figure 4-8. Interface Diagrams

For Devices, the read or write operation is the same in order to be able to send status information back to the interpreter. The MDC controls the lock and lock checking before the operation is enabled. The Device interface contains a counter used to count incoming HS clock pulses and initiate the read or write. When the operation is finished, high speed clocks (DRC) are gated back to the MDC and counted there to determine when the transfer is completed.

4.3.3 Memory Request Control

There exist two possibilities to consider when an interpreter requests access to a memory module: The interpreter was the last one to use that memory, and the interpreter was not the last one to use that memory. In the latter situation the following series of events occurs in the SWI.

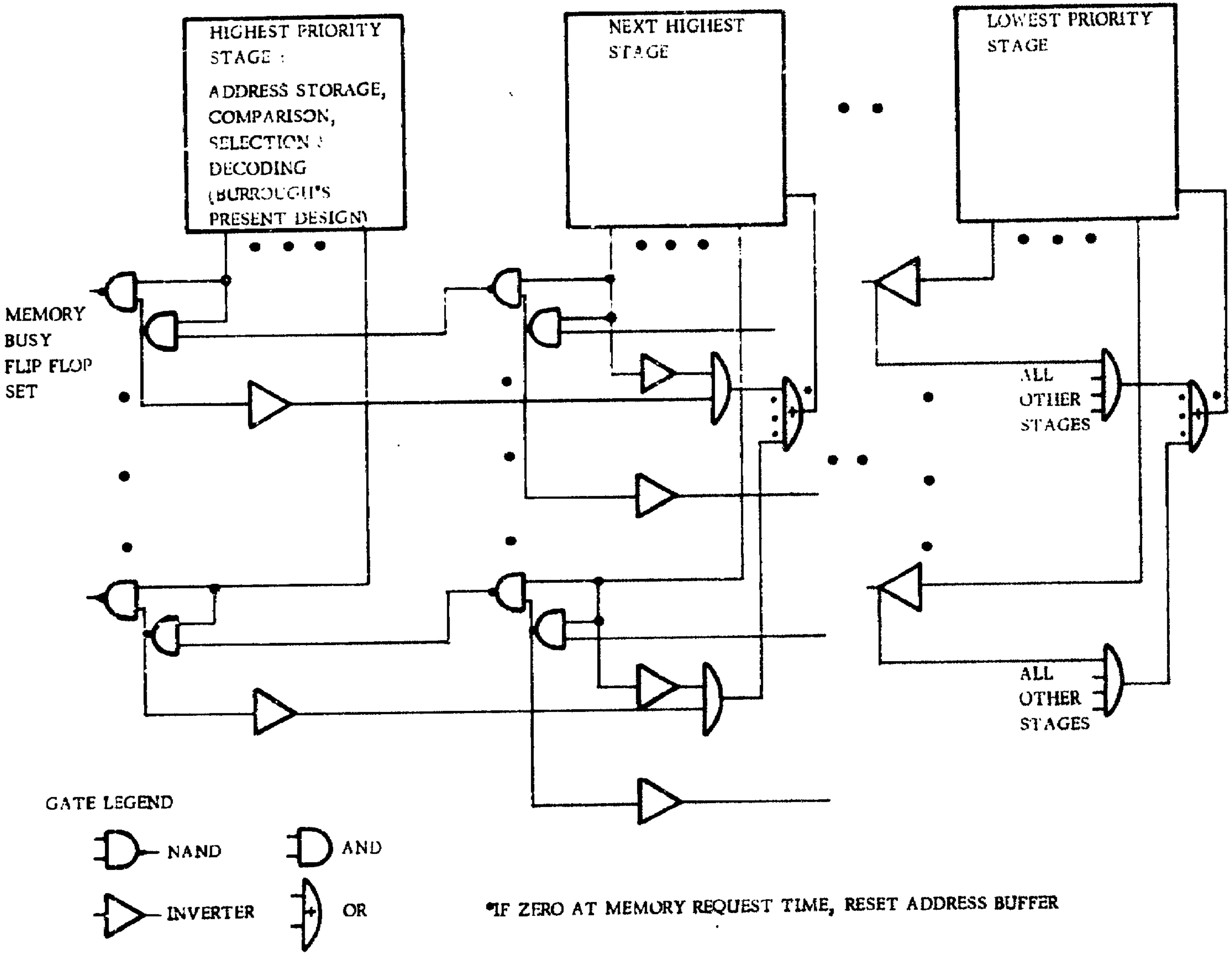
The comparison logic with MC is inhibited by the Valid Enable not being true. The Memory Request (MR) then gates the 3 bit memory module address to test whether the memory module of interest is busy.

1. It is busy leads to continuing inhibit of the interpreter clock (HS/5). The denial is ended by the addressed memory clearing the busy Flip Flop (FF) with a cycle complete. The highest priority interpreter is permitted to gate the Interpreter Clock (Int Clk). The others are locked out.

2. It is not busy results in the Int Clk enabled to transfer the address into storage and set the Valid Address FF. A comparison is made and a pulse generated which sets the busy FF of that addressed memory module. This setting is sensed by the Use Determination logic and a reset generated to clear the address registers of any other Memory Control (MC) channel holding that memory module address.

In the former case with the interpreter having last addressed the requested memory module, the address register in the MC comparison logic will contain the same three bits. The memory request will then be immediately granted. In implementing this case Burroughs ran into problems of conflicts when different requests came into the SWI at slightly different times. In some cases access would be granted to two interpreters to the same memory module. This problem was eliminated by disabling the comparison logic that grants immediate access to an interpreter if it last used the same memory module. As a result of this all memory requests must go through the priority logic with the resultant added delays in granting access to an interpreter.

It is possible to structure the logic in the MC module to permit the operation to be as originally desired. Figure 4-9 shows an approach and provides the means by which the quantity of logic can be estimated. The addressed memory module is first tested for its busy status. If available, the memory request enables the comparison pulse to be generated which initiates the memory operation. Simultaneous requests from two interpreters results in an inhibit signal being generated in lower priority MC's to prevent enabling the memory request gating.



*IF ZERO AT MEMORY REQUEST TIME, RESET ADDRESS BUFFER

Figure 4-9. MC Logic Revision

4.3.4 SWI Design Features

Some of the features designed into the SWI are noted below. The memory cycle and device cycle times can be variable due to the asynchronous interface allowed by the SWI. This allows a mixture of memory types to be used for the main memory of the system.

One word transfers are only possible. For each word transferred, the interpreter must issue a new request. This holds true for memories and devices.

Modularity was to be one of the key design features of the SWI. It is felt that modularity can be achieved in the design of the SWI. However, in investigating the modularity of the present SWI design it became apparent that a very limited degree of modularity exists. A discussion of the modularity achieved in the present design is given below.

The MDC is modular in that one is used per interpreter. There is no apparent limitation as to how many MDC's could be used.

The MC is made up of two types of modules. The 'MCI' module handles two interpreters and the Memory Busy FF's. The 'MCO' module handles three interpreters. MCI could be used alone, if only one or two interpreters were in the system, except that pull up resistors are required which are contained on the MCO module. It appears the concept could handle more than one MCO module as long as circuit limitations are not exceeded. This would allow more than five interpreters to be used. The MC modules handle eight memories. There is no modularity as far as the number of memories accommodated by the MC. The same module is used for one or for eight memories. The present design of the MC cannot easily be modified to service more than eight memories. One possible approach to service more than eight memories is to add another MC module to handle another bank of eight memories. This would require some means of handling an extra address bit to signify which bank of eight memories to use; such a modification would have to be added either to the MDC or the MC.

The DC is made up of two modules. Both modules are identical and handle up to three interpreters by eight memories. One module may be used alone if three or less interpreters are to be serviced. It appears that more than six interpreters can be serviced by using more than two DC modules if the circuit, limitations are not exceeded. Expansion beyond eight device modules is not possible with the DC. Handling more than eight devices would require modifications to the MDC or DC as discussed above for the MC.

The Output Switch Network uses two types of modules, OSNI and OSNO. The OSNI handles two bits and one clock per interpreter and the OSNO handles four bits per interpreter. Each type services five interpreters by eight memories. The address output requires two OSNI modules since two clocks need to be transmitted (clear pulse and high speed clock). If more bits are desired to be transmitted in parallel, then an OSNO module can be added (pending circuit limitations). This would add four more bits to be transmitted in parallel. Additional OSNO modules would increase the parallelism in transmission. The data output uses two OSNO modules. Additional OSNO modules would increase the parallelism in transmission providing circuit limitations are not exceeded. It appears expansion beyond five interpreters is not possible since the clock bits cannot be noded together from two OSNO modules in parallel. The OSNI modules can be noded together (pending circuit limitations) to handle more than five interpreters. More than eight memory modules can be serviced by using additional OSN modules in parallel provided the MC modules are modified to handle more than eight memory modules.

The Input Switch Network uses one type of module that provides transmission of four bits per interpreter and services five interpreters by eight memories. Two modules are used in the system providing transmission of eight bits in parallel. Modules may be added to increase the parallelism, provided circuit limitations are not exceeded. Likewise, more than five interpreters could be serviced by adding modules in parallel provided circuit limitations are not exceeded. Expansion beyond eight memory modules is also possible provided the MC modules are modified to handle more than eight memory modules.

Another consideration in the SWI that is tied in with modularity is the failure tolerance aspects of the SWI. The existing SWI design contains a number of failure points which can cause loss of more than one element or cause difficulty in detection or reconfiguration. After considering each of the SWI elements, some possible failure modes are noted below.

1. DC - The most obvious DC feature contributing to lower reliability and to difficulties in detection and reconfiguration is the use of common logic between stages. Some failures such as those contributing to the generation of MDC control signals can occur. These can be isolated and confined to one stage if the power connections permit. However, others that are described next use common logic and cannot be.

   a. Common address selection is used for up to three stages. The address select can fail and prevent the ability to address decode. The entire SWI can be lost.
   b. The "OR" gate providing lock inhibit signals is used for three stages and noded with the others from the second DC for more-than-three interpreter systems. Its loss would cause loss of a particular device. Loss of the source of power for this common "OR" logic would mean loss of more than one stage or even total loss of the SWI.
   c. The priority inhibit logic is used between stages. A failure in this area would be hard to isolate and reconfigure around.

2. MC - As with the other SWI elements, certain MC failures such as the comparator can be isolated to one interpreter. Other types of failures which can occur with the MC and which present greater problems are as follows:

   a. Lower stage lockout - The priority signals are cascaded. A failure of the signal or a power turnoff to an intermediate stage can lock out lower stages.
   b. Multiple memory accessing - Inability to set the busy flip flop can cause two interpreters to request access to one memory. Also if the flip flop for reset of the address buffer fails, two interpreters can also access a memory.
   c. Failure to transfer information - A busy flip flop "ON" failure can prevent other channels from accessing a memory. Also, a single power source provides power to these flip flops making it difficult to turn power off to a failed stage.

3. ISN/OSN - The design of the ISN/OSN is such that single failures for the most part contribute to loss of either an interpreter, memory, or device.

   Failure of the strobe signal to "ON" can cause the continuous readout of whatever is addressed by the buffer storage for that stage. Since separate power is supplied to each channel, it is possible to remove the faulty channel and lose its associated interpreter.

   Inputs to interpreters from devices through ISN's are tied to those from memories. Loss of one line prevents that interpreter from communicating with either. Isolation and power removal to the bad component may restore the communications over the good lines.

4.3.5 Alternate SWI Design

From the above discussions, it is obvious the modularity and failure tolerance aspects of the SWI need to be investigated further. In considering alternate designs, it is necessary to examine the partitioning of the electronics to the boards and the number of pins required for interconnecting the SWI. It is desirable to have flexibility for different word sizes; number of interpreters, memories, or devices; to make all channels alike to achieve submodularity, and to minimize the board types. Achieving all of these is difficult due to packaging limitations arising from the need to minimize circuit connectors used, the number of pins and board sizes which are available, and to utilize available standard solid state devices. The following represents an approach which offers greater potential in meeting the avionics needs than the present approach. (The approach will follow the same commercial prototype packaging philosophy as Burroughs used in the multiprocessor, enabling a one-to-one comparison. An avionics design may take on a slightly different approach.)

Separation of the channels and integration of all logic relative to a channel is suggested. A channel is hereby defined as the information path for one interpreter. The MDC design can remain as is since it is designed for a single interpreter or channel approach. The circuitry for the MC and DC can be packaged on one board. The ISN/OSN can either remain as presently designed or redesigned with each channel on one board and containing the circuitry for all or portions of an ISN, data OSN and address OSN. This latter approach requires junctions external to the board in order to enable inputs to go-to each ISN (as contrasted to these junctions being presently provided on the circuit boards). Similarly all corresponding OSN outputs would have to be tied externally to enable any of the outputs to go to the proper device or memory.

The alternate designs achieving this channel modularity and failure tolerance on a channel basis are shown in Figures 4-10, 4-11 and 4-12. The MC and DC channel "slices" are shown in Figures 4-10 and 4-11. These two can be packaged on one board, hereafter referred to as the MC/DC module. The MC/DC module provides all the memory and device control for one interpreter. The former ISN, OSN-0, and OSN-1 modules are changed into one module now called the IOSN. Figure 4-12 shows a dual channel version of the IOSN, a triple channel version is also feasible with the commercial prototype technology. Each channel on the IOSN module provides four data bits and one clock bit out, two address bits out, and four data bits and one clock bit in (to/from up to eight memory modules or eight devices).

A comparison with the previous design features shows:

1. No common logic exists between stages. Reconfiguration and power turnoff is possible.

2. Error detection logic such as parity bit can be added.

3. Modularity by stages is achieved and provides advantages in checking and interchangeability.

The previous design consisted of a large number of boards and types of boards as shown in Table 4-5.

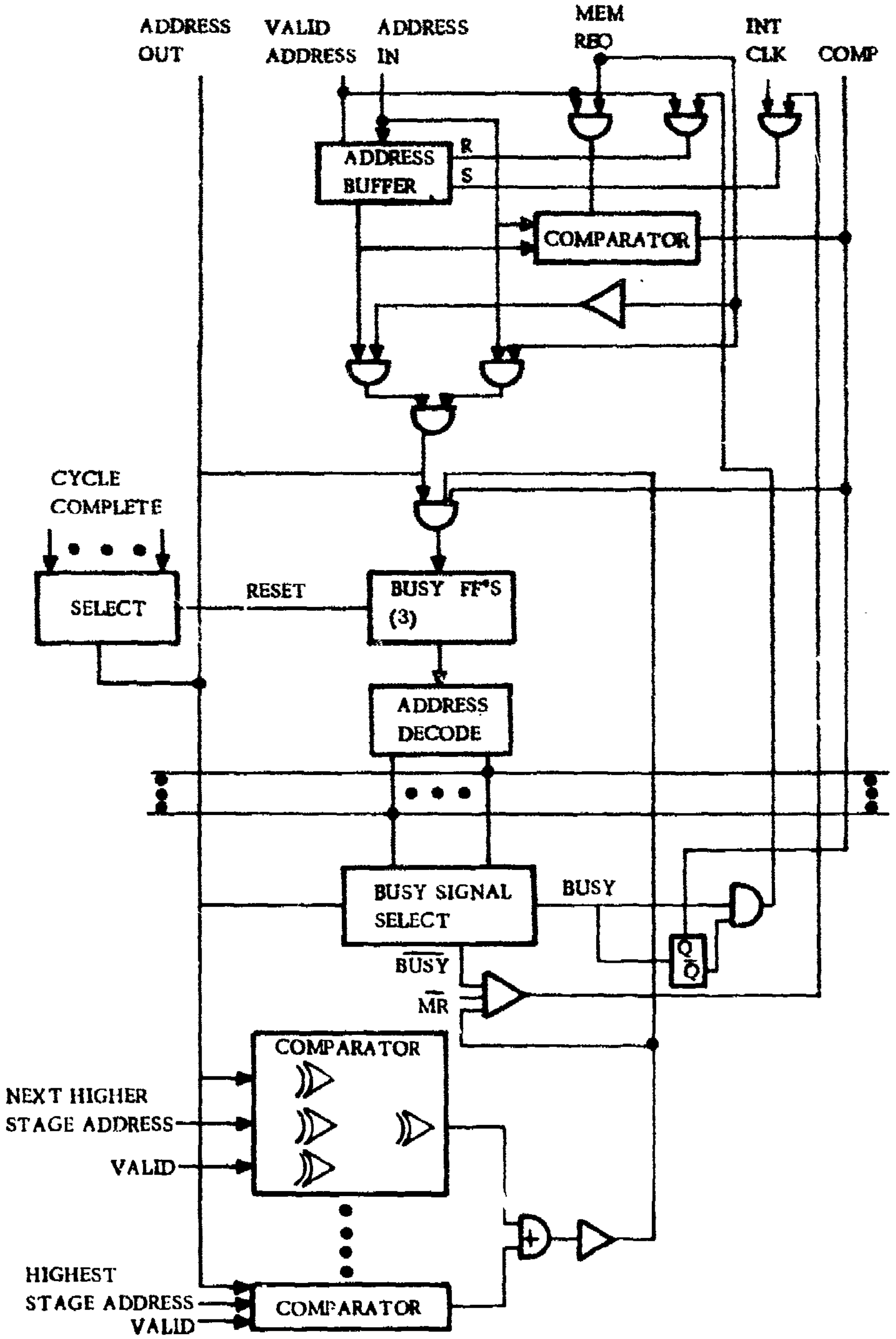


Figure 4-10. Typical Stage – Memory Control, Alternate

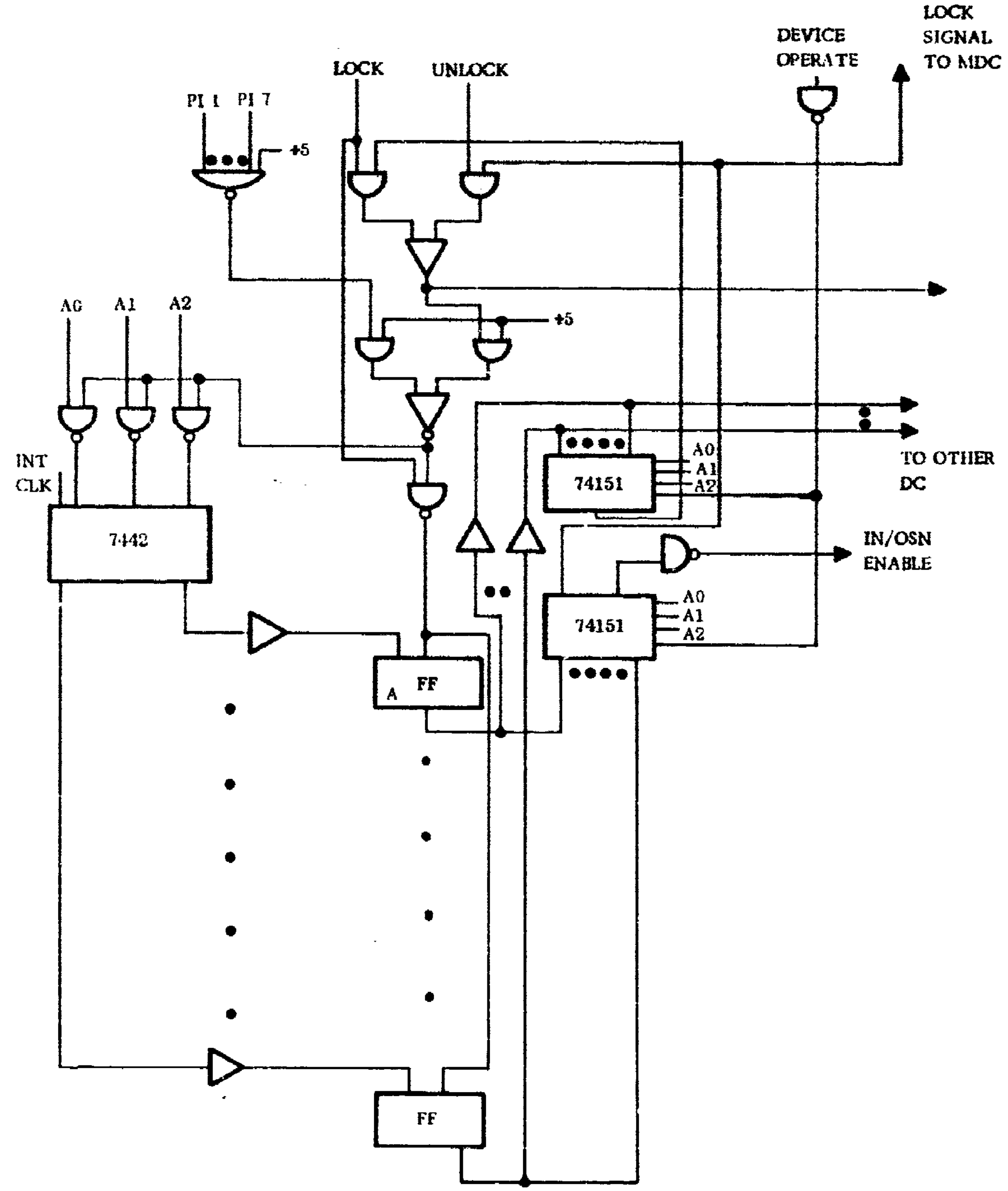
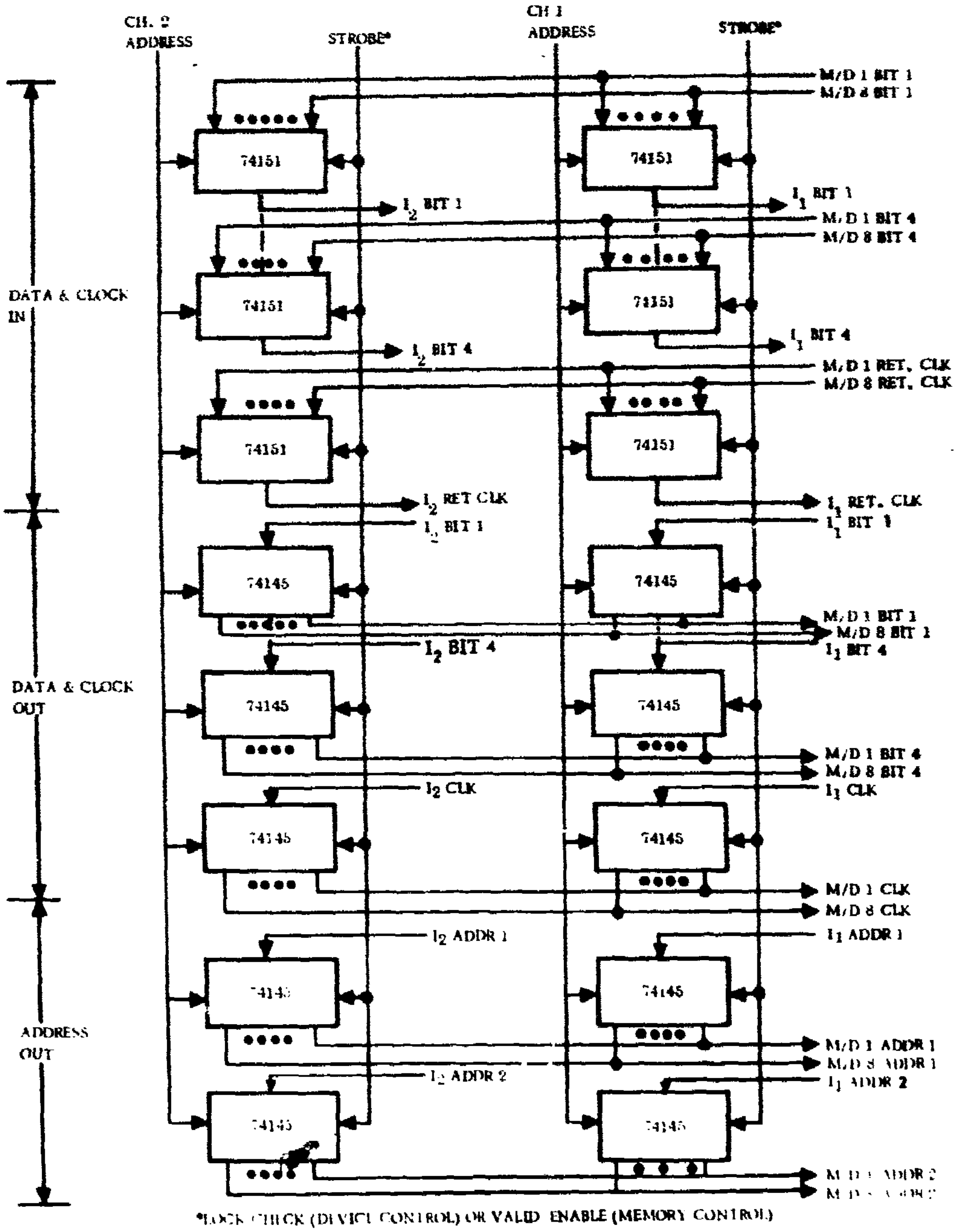


Figure 4-11. Typical Stage – Device Control, Alternate



*LOCK CHECK (DEVICE CONTROL) OR VALID ENABLE (MEMORY CONTROL)

Figure 4-12. Input/Output Switch Network (IOSN) – Dual Channel

Table 4-5. Number of Boards in Present SWI

SWI Element Int. Qty	ISN	OSN Data	OSN ADDR	MC	DC	MDC	Total
	Per MC or DC						
2	2	3	1	1	1	2	16
3	2	3	1	1	1	3	17
4	2	3	1	2	2	4	20
5	2	3	1	2	2	5	21
Types	1	2	-	2	1	1	7

Table 4-6 presents the number of boards for different numbers of interpreters (assuming a two channel IOSN and a three channel IOSN) for the alternate design approach.

Table 4-6. Number of Boards for Alternate Design

SWI Element Int. Qty	IOSN	MC/DC	MDC	Total
	Per MC/DC			
2	2	2	2	8
3	2	3	3	10
4	4	4	4	16
5	4	5	5	18
Types	2	1	1	4

This approach provides a better utilization of the electronics in matching the number of interpreters with the number of SWI boards.

The MC is shown to be similar to the present design. Each stage has three busy flip flops settable by the compare pulse when enabled by the priority sensing. Each stage would have a network to select the correct cycle complete signal to reset its busy flipflops. Each stage also must have comparators which compare the stage's address with any of higher priority to generate this priority enable. For a five interpreter capability, a maximum of four comparators is needed for the lowest stage. It would be practical to make all stages alike and tie off any unused stages. Since all eight combinations of address are used, the valid signal for each stage must also be used to enable the comparison. Further, if the higher stage is turned off, the comparison section for that stage must be so designed that its output cannot occur.

The MC operation is similar to the present.

1. New address - First thing to occur is the check of the busy signals. If OK, the address is dropped into the buffer and a comparison signal generated. This sets the busy flip flop and initiates the operation.

2. Same address - A comparison is made immediately. An inhibit is also generated with any lower address until the busy signal logic can generate a reset for that logic. This new logic overcomes the previous described problem of dual accessing presented in Section 4.3.3.

Without the ability, as described in Section 4.3.3, to permit the same interpreter to use the same memory, an additional three integrated circuits per MC stage would be needed for this new design over the present. With the function added to the original design, this approach uses about six less integrated circuits per stage than the approach shown in Section 4.3.3.

To accommodate the interconnections about 50 pins are required.

The DC logic increases slightly with the elimination of common logic. About three more integrated circuits per stage are required to perform the same functions. The number of additional pins per stage needed is about 15.

The conclusion reached from the considerations of quantities of integrated circuits and interconnection pins is that it is practical to place the MC and DC logic per stage on one board. This is so indicated in the previous table as the MC/DC module.

Since individual channels were recognized, the IO design follows that of the present except that a different packaging is recommended as a means of reducing both types and quantities of boards. A number of ways that the input output area may be structured were investigated, including single, dual, and triple channel. The recommended approach is that of providing two board types. One board contains half of the IO logic (for eight memories or eight devices) for two channels and the other board half for three channels. This gives better utilization of the electronics when the number of interpreters is odd and while keeping the total board count down. The estimated interconnection pins and integrated circuits per board support the conclusion that these approaches are compatible with the board size and connector available (150 pin connector and 45-16 pin integrated circuits per board).

The resulting Switch Interlock is shown in Figure 4-13. All logic associated with a channel could be connected to the power source regulator for that channel. This design provides better modularity and reconfigurability than the existing approach.

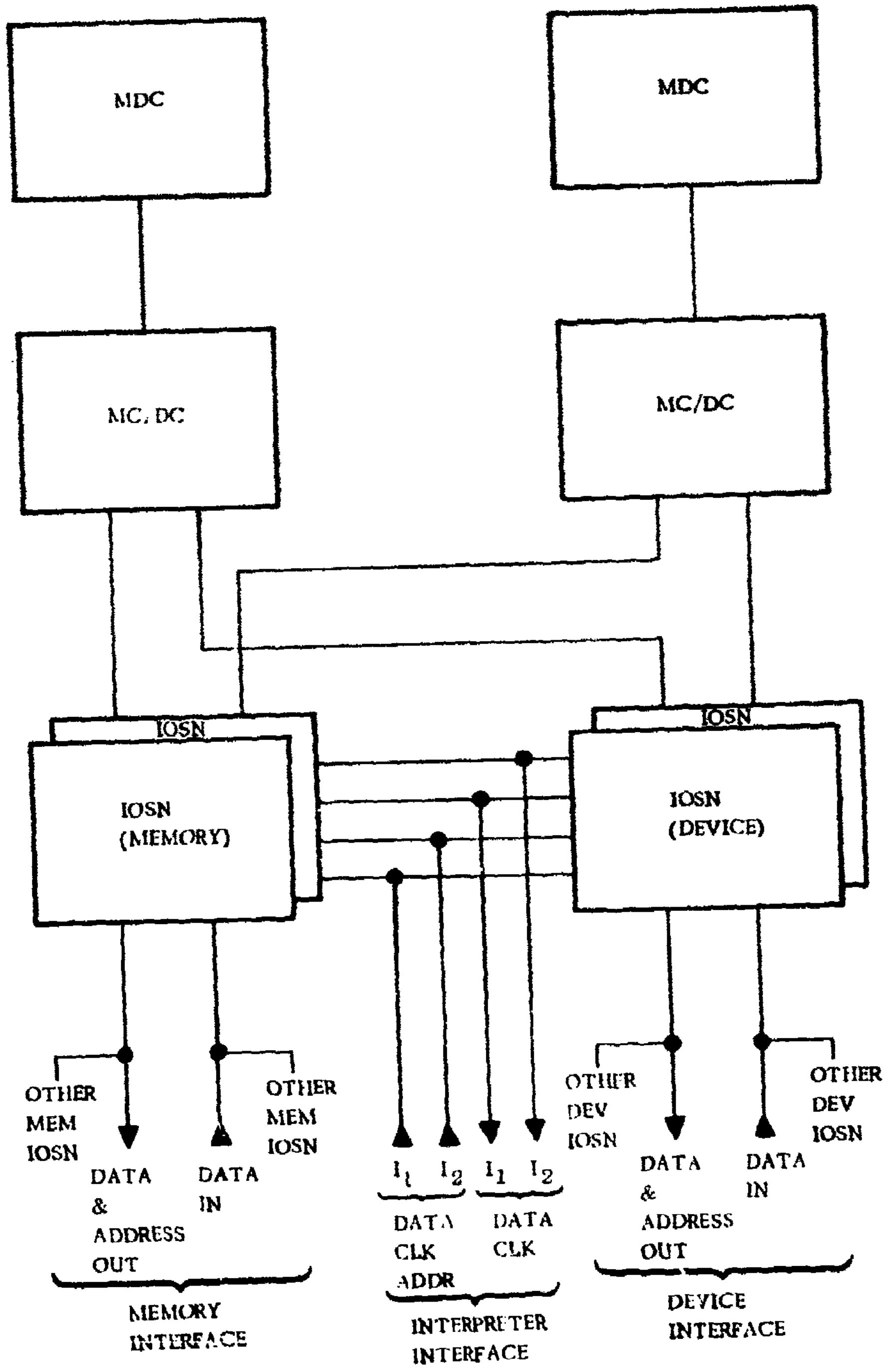


Figure 4-13. Alternate Switch Interlock for a Two Interpreter System (Two Channel IOSN)

4.4 MULTIPROCESSOR PERFORMANCE CAPABILITY

4.4.1 Introduction

Section 4.1 discussed the flexibility inherent in the multiprocessor and identified different approaches to the S language. It is the intent of this section to define the performance capability, quantitatively, in terms of its throughput or speed. To accomplish this the emulation mode of operation will be investigated further. The emulation of the IBM 4π CP avionics computer was investigated. In addition such an emulation modified by the use of macros was investigated. This gave the resultant throughput improvement in using macros and also an estimate of storage redrction. This activity also enabled an estimate to be made of the amount of MPM and NM required.

4.4.2 Emulation Mode of Operation

The IBM 4π CP avionics computer was selected for the emulation application. This computer is characterized as a 16/32 bit (instruction and data) machine with the following primary registers:

32 bit accumulator (A)

32 bit lower accumulator (Q)

16 bit instruction counter

3-16 bit Index Registers

The instruction formats are as follows:

1/2 Word —

5	1	2	8
OP Code	H/F	T	Displacement

T — index registers

H/F — 0: 1/2 word instruction

Full word —

5	1	2	1	3	4	16
OP Code	H/F	T	I_A	000	OP Code ext	Address

I_A — 0: direct addressing

I_A — 1: indirect addressing

In emulating this machine one of the A registers, A_1, was dedicated as the program counter, and A_2 was dedicated as the accumulator, the remaining registers were stored in main memory.

There are two basic types of instructions to be considered: 1/2 word (16 bits) and full word (32 bits). Within the 1/2 word format there are thirty-one instructions. Seventeen of these use the regular format where the T bits are used in a normal manner to generate an effective operand address. The remaining instructions utilize the bits in various ways. For example, the shift instructions use two op codes but utilize the T bits in specifying the type of shift and part of the displacement to specify the shift amount. Similar distinct uses of the instruction bits occur for several other instructions.

Within the full word format there are 27 instructions that each require special functions to be performed with the bits in the instruction, some do not require interpretation of the op code ext, others do not use the effective address, others require interpretation of the T bits, etc.

The operation of the interpreter in this emulation mode is illustrated in Figure 4-14.

There exist three basic steps in the emulation process: instruction fetch, instruction format interpretation, and instruction execution. In many cases (certain instructions) the instruction format interpretation is integral with the instruction execution and is not necessarily a separate routine for certain instructions as illustrated in Figure 4-14. These concepts are further illustrated in Figure 4-15 where the structure of these routines is shown. The concept is that the I fetch routine uses the op code to access a table of op code pointers (1) these pointers either lead to the I format

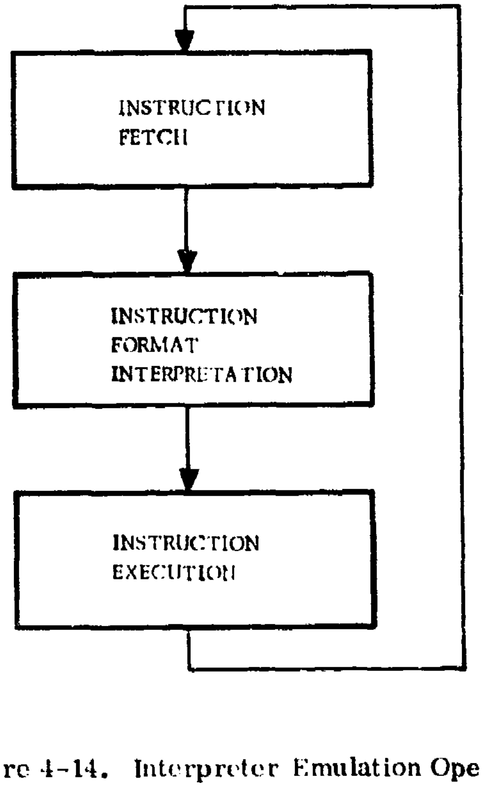


Figure 4-14. Interpreter Emulation Operation

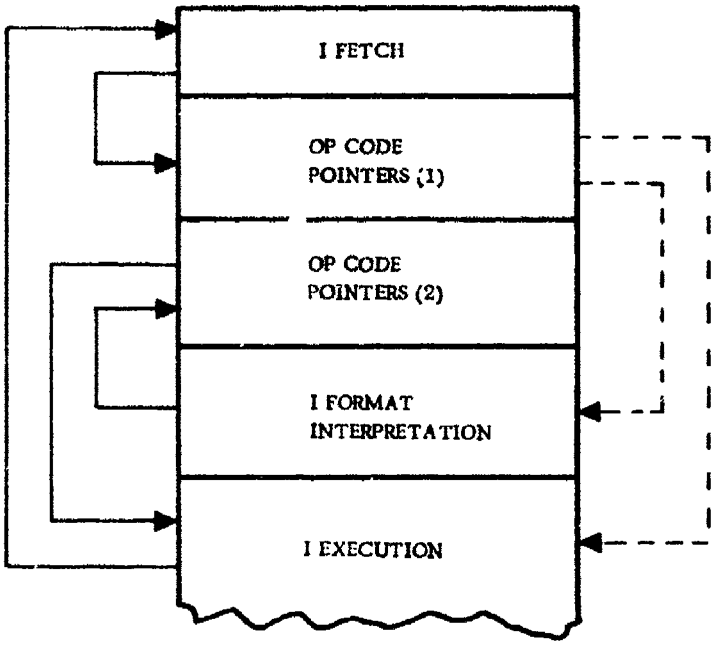


Figure 4-15. MPM Structure

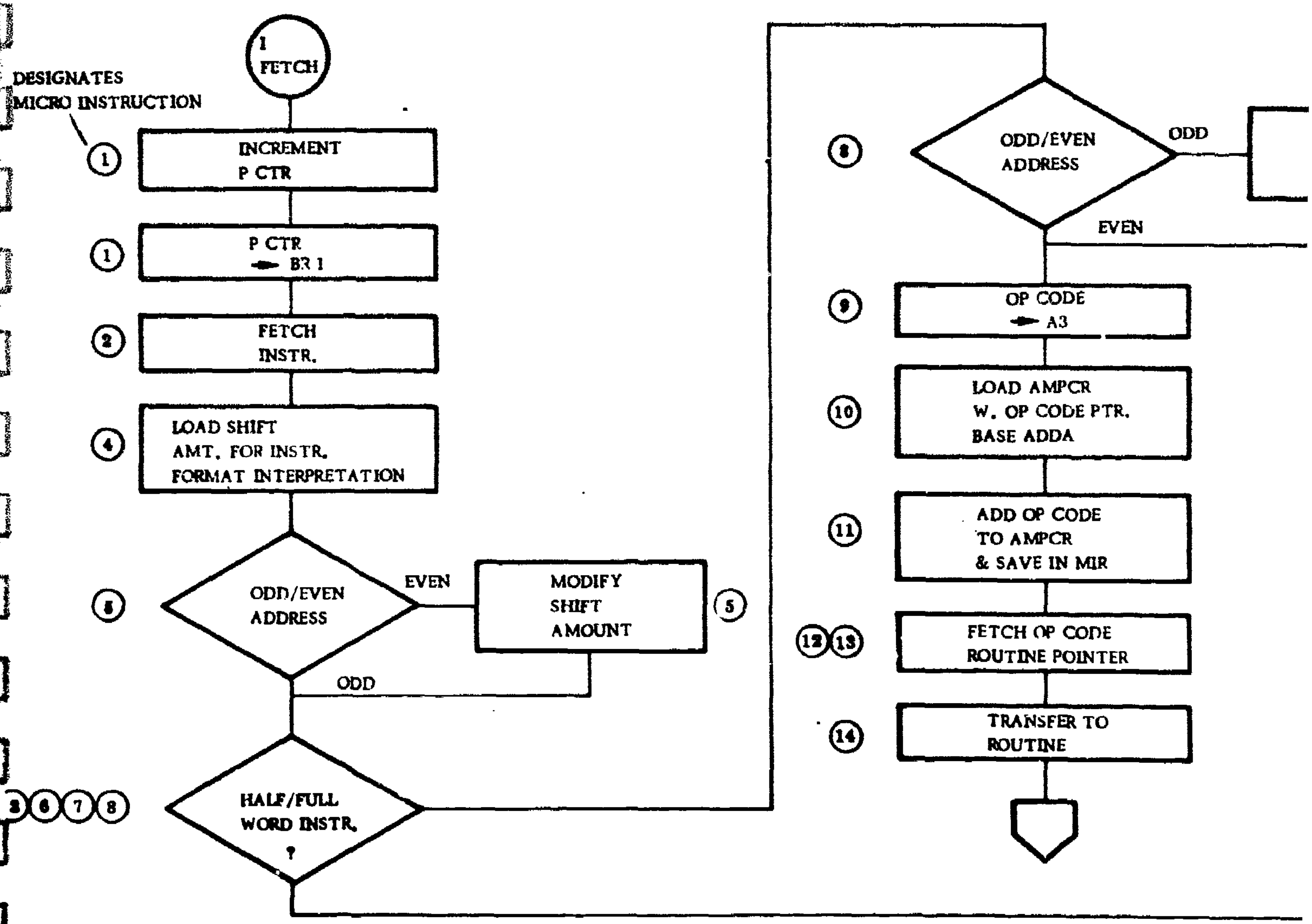
interpretation or to the I execution routines. If the I format interpretation routine was entered, it will point to a set of op code pointers (2) that will then point to the appropriate I execution routine. At the completion of an I execution routine, a pointer is used to re-enter the I fetch routine.

The I fetch routine flow chart is shown in Figure 4-16 and the detailed microcoding is given in Table 4-7. The flow charts along with the microcoding table are for the most part self explanatory.

The flow chart for the instruction format interpretation routine is shown in Figure 4-17 and the detailed microcoding in Table 4-8. It should be noted that this routine is entered with the instruction in the B register aligned as follows:

	2	8	16	5	1
bits:	T	Displ	- - - - - - - - - - - - - -	op code	H/F

An additional point to keep in mind in this emulation process is that the interpreter logic unit and its interface with the main memory is 32 bits. The emulated machine also has a 32 bit logic unit and memory interface. As a result, it must be kept in mind which half of the word (instruction and data) is being processed and also if a full word instruction is being processed whether the entire instruction has been received.


1 FETCH
DESIGNATES MICRO INSTRUCTION
1
INCREMENT P CTR
1
P CTR → BR 1
2
FETCH INSTR.
4
LOAD SHIFT AMT. FOR INSTR. FORMAT INTERPRETATION
5
ODD/EVEN ADDRESS
EVEN
MODIFY SHIFT AMOUNT
5
ODD
3 6 7 8
HALF/FULL WORD INSTR. ?
8
ODD/EVEN ADDRESS
ODD
EVEN
9
OP CODE → A3
10
LOAD AMPCR W. OP CODE PTR. BASE ADDA
11
ADD OP CODE TO AMPCR & SAVE IN MIR
12 13
FETCH OP CODE ROUTINE POINTER
14
TRANSFER TO ROUTINE

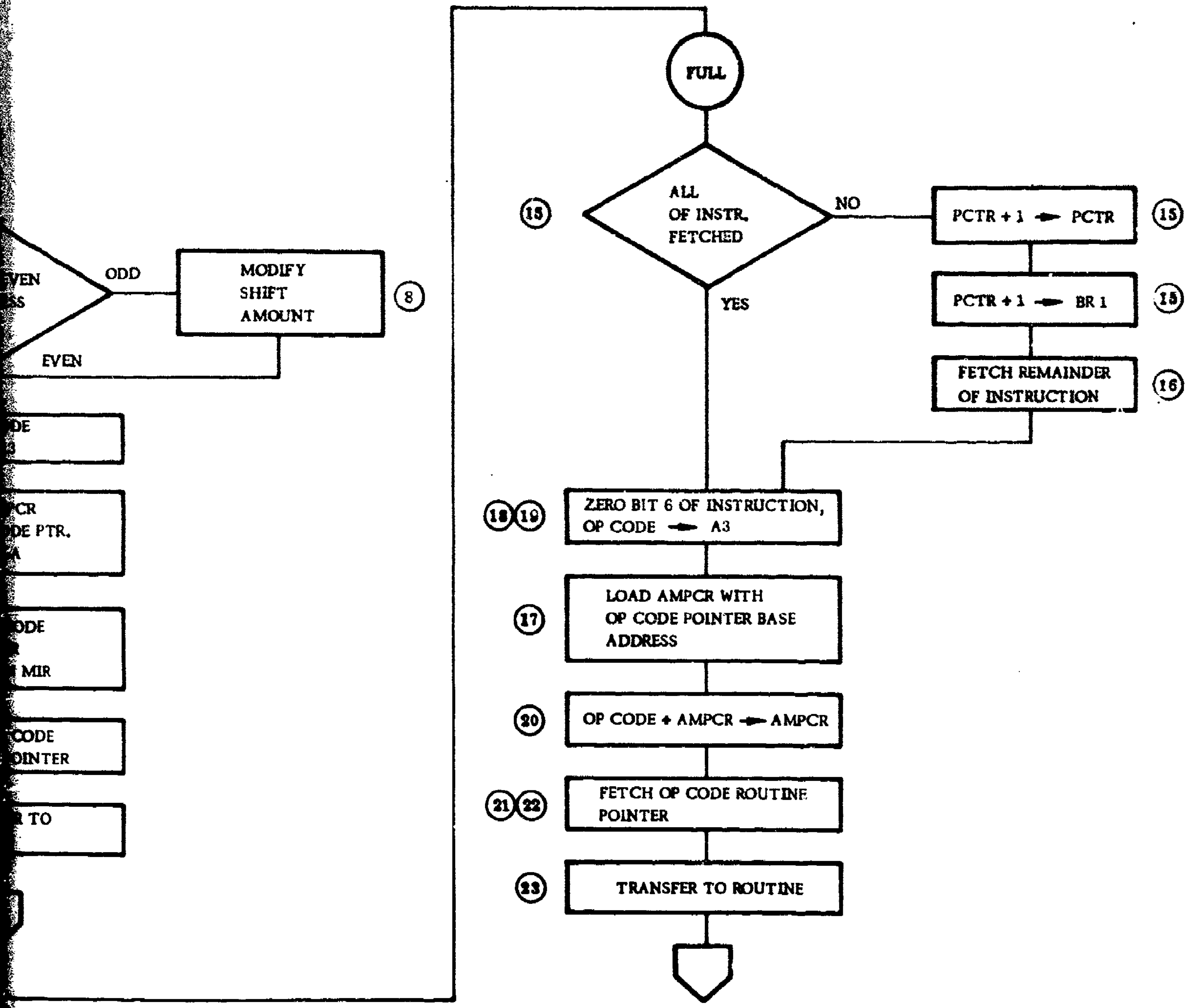


Figure 4-16. I Fetch Rout'ne

Table 4-7. Microcoding for I Fetch

Microinstr	Microcode	Comments
1	LU uncond, SET LC2 IF AOV, A_1 → A, B 100 → B, Add A&B, BSW → A1 & BR1	
2	Ext Op Cond, SC = RDC, MR1, LU uncond, A1 → A, B000 → B, Add A&B	Makes sure Mem op is complete from I execution, need to repeat A1 through adder for dynamic conditions to remain present in 5
3	Type 2 Load AMPCR	With address to full word part of routine
4	Type 2 Load LIT w.26, SAR w 10	
5	LU Cond, $\overline{SC}$ = LST, LIT → Z, B000 → B, Add Z & B, BSW → SAR, SET LC 1 if LST	LC 1 SET IF ODD ADDRESS in A1
6	LU Cond, SC = RDC, SC = 0 then wait, BEX	Mem request in 2 needed
7	LU uncond, Rt Shift B, BSW → B	
8	LU Cond, $\overline{SC}$ = MST, $\overline{SC}$ = 0 then Jump, LIT → Z, B000 → B, Add B&Z, BSW → SAR	Jump to full word part of I fetch if bit 6 of instruction is 1
9	LU uncond, Left Shift B, BSW → A3	Only op code is in A3
10	Type 2 Load AMPCR	With op code pointer base address
11	LU uncond, A3 → A, AMPCR → Z, Add A&Z, BSW → AMPCR & MIR	
12	LU uncond, Successor execute	
13	Type 2 Load AMPCR	One of these per op code, points to I Format or I Execution
14	LU uncond, Successor Jump	
15	LU cond, SC = LC 1, A1 → A, B 100 → B, ADD A&B, BSW → A1 & BR1, if SC = 0 then skip	

Table 4-7. (Cont)

Microinstr	Microcode	Comments
16	Ext Op Uncond, MR1	
17	Type 2 Load AMPCR	With op code pointer base address
18	Type 2 Load SAR with 26	
19	LU uncond, O → A, BTTO → B, Add A & B, Left Shift 26, BSW → A3	Only op code is in A3
20	LU uncond, A3 → A, AMPCR → Z, Add A & Z, BSW → AMPCR	
21	LU uncond, Successor execute	
22	Type 2 Load AMPCR	1 location per op code
23	LU uncond, Successor jump	

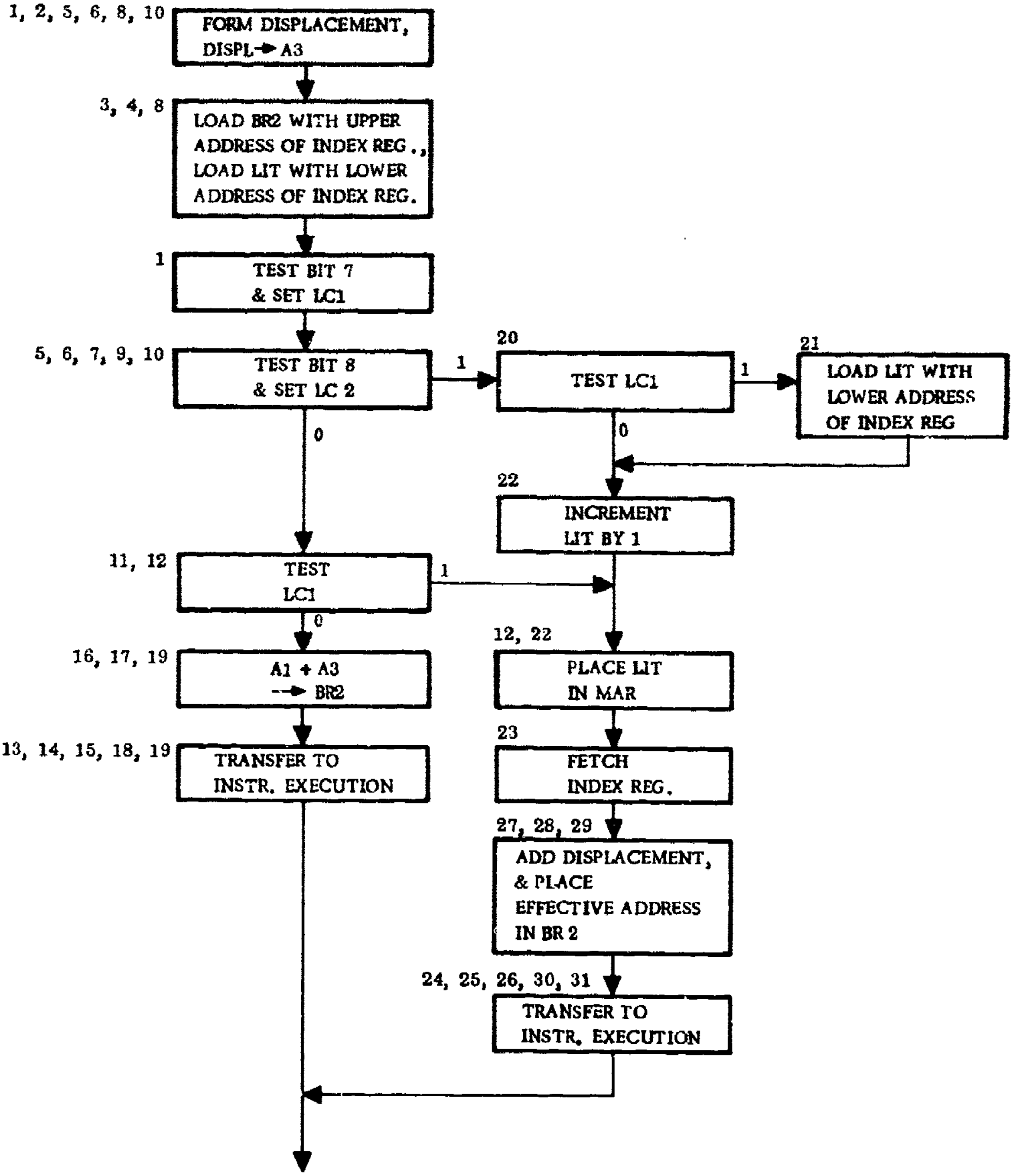

1, 2, 5, 6, 8, 10
FORM DISPLACEMENT, DISPL→A3
3, 4, 8
LOAD BR2 WITH UPPER ADDRESS OF INDEX REG., LOAD LIT WITH LOWER ADDRESS OF INDEX REG.
1
TEST BIT 7 & SET LC1
5, 6, 7, 9, 10
TEST BIT 8 & SET LC 2
1
20
TEST LC1
1
21
LOAD LIT WITH LOWER ADDRESS OF INDEX REG
0
0
22
INCREMENT LIT BY 1
11, 12
TEST LC1
1
0
12, 22
PLACE LIT IN MAR
16, 17, 19
A1 + A3 → BR2
13, 14, 15, 18, 19
TRANSFER TO INSTR. EXECUTION
23
FETCH INDEX REG.
27, 28, 29
ADD DISPLACEMENT, & PLACE EFFECTIVE ADDRESS IN BR 2
24, 25, 26, 30, 31
TRANSFER TO INSTR. EXECUTION

Figure 4-17. Instruction Format Interpretation Routine

Table 4-8. Microcoding for Instruction Format Interpretation

Microinstr	Microcode	Comments
1	Type 2, Load SAR with 22	
2	LU uncond, SC = LST, SC = 1 then set LC 1, Right Shift B, BSW → A3	
3	Type 2, Load Lit	
4	LU uncond, Lit → Z, B000 → B, Add Z & B, BSW → BR2	
5	Type 2, Load SAR with 1	
6	LU uncond, Left Shift B	
7	LU uncond, SC = LST, SC = 1 the set LC 2	
8	Type 2, Load Lit, Load SAR with 24	
9	Type 2, Load AMPCR	With jump address if bit 8 = 1
10	LU uncond, SC = LC 2, SC = 1 then jump otherwise step, Left Shift A3, BSW → A3	*Displacement* → A3 *jump is to* 20
11	Type 2 Load AMPCR	
12	LU cond, SC = LC 1, SC = 1 then jump otherwise step, Lit → Z, *B000* → B, *Add B&Z*, BSW → MAR	Jump is to 23
13	Type 2, Load Lit	With increment to execution routine pointer

Table 4-8. (Cont)

Microinstr	Microcode	Comments
14	LU uncond, MIR → B	
15	LU uncond, Lit → Z, BTTT → B, Add B&Z, BSW → AMPCR	
16	LU uncond, A3 → A, B000 → B, Add A&B, BSW → B	
17	LU uncond, A1 → A, BTTT → B, Add A&B, BSW → BR2, Successor execute	
18	Type 2, Load AMPCR	
19	LU uncond, if SC = LST then set LC1, Successor jump	
20	LU uncond, SC = LC1, if SC = 1 then step otherwise skip	
21	Type 2, Load Lit	
22	LU uncond, Lit → Z, B100 → B, Add B&Z, BSW → MAR	
23	Ext Op uncond, MR2 , SC = LST, SC = 1 then set LC1	
24	Type 2 Load Lit, Load SAR with 16	
25	LU uncond, MIR → B	
26	LU uncond, Lit → Z, BTTT → B, Add B&Z, BSW → AMPCR	
27	LU cond, SC = RDC, SC = 0 then wait, BEX	
28	LU cond, SC = LC1, Left Shift B, BSW → B	
29	LU uncond, A3 → A, BTTT → B, Add A&B, BSW → BR2 , Successor execute	
30	Type 2, Load AMPCR	
31	LU uncond, SC = LST, SC = 1 then set LC1, Successor Jump	

Most of the IBM 4π CP instruction set was microprogrammed. It was found that most of the short (store, load, add, compare, etc) type of instructions take a similar amount of time to execute. Therefore, several examples will be given below.

The flow chart for the full word (32 bits data) add instruction using a 1/2 word (16 bits) instruction format is shown in Figure 4-18 and the corresponding microcode is given in Table 4-9. It is seen that the execution portion is relatively simple compared to the I fetch and I format interpretation. The 1/2 word load and full word compare using a 1/2 word instruction format are given in Figures 4-19 and 4-20 and Tables 4-10 and 4-11. A flow chart for the multiply routine is given in Figure 4-21. The microcoding for this routine is not given since it is identical to that given by Burroughs in Reference 1. These instructions were branched to the I execution phase from the I format interpretation routine as shown in Figure 4-17. Most of the short format instructions are executed in this manner. However, some are entered directly from the I fetch routine, such as the shift instructions. It is expected that a similar amount of total time will also be required by these instructions.

A flow chart for a full word format instruction is shown in Figure 4-22. The particular instruction is a 1/2 word add with the microcoding given in Table 4-12. This flow chart is entered directly from I fetch and therefore includes the I format interpretation and execution. At this point in time it is not certain whether each full word format instruction will have its own I format and execution routine or whether a common I format interpretation routine can be used as for the 1/2 word format instructions.

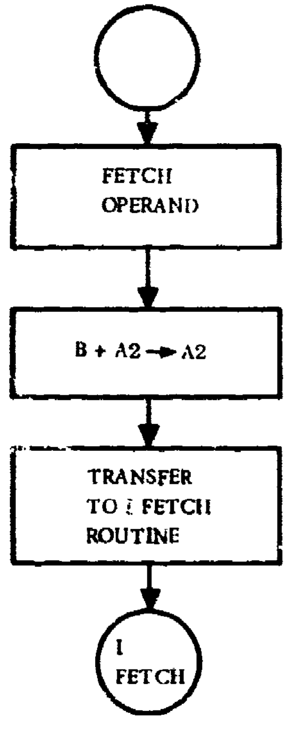


Figure 4-18. Full Word Add Instruction Execution (1/2 Word Instruction Format)

Table 4-9. Microcode for Full Word Add Instruction Execution

Microinstr	Microcode	Comments
1	Ext Op uncond, MR2	
2	Type 2, Load AMPCR	With jump address to I fetch
3	LU cond, SC = RDC, SC = 0 then wait, BEX	
4	LU uncond, A2 → A, BTTT → B, Add A&B, BSW → A2, Successor = Jump	

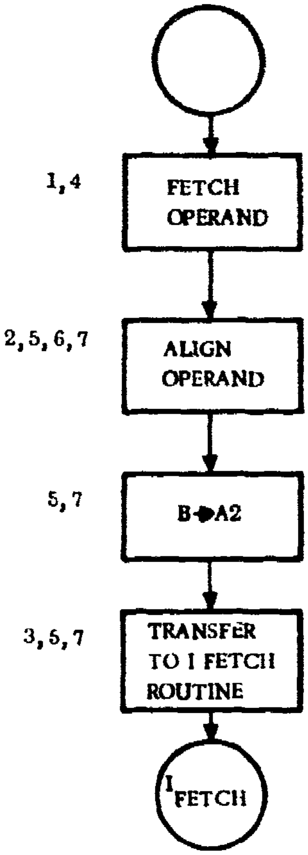


Figure 4-19. Half Word Load Instruction Execution
(1/2 Word Instruction Format)

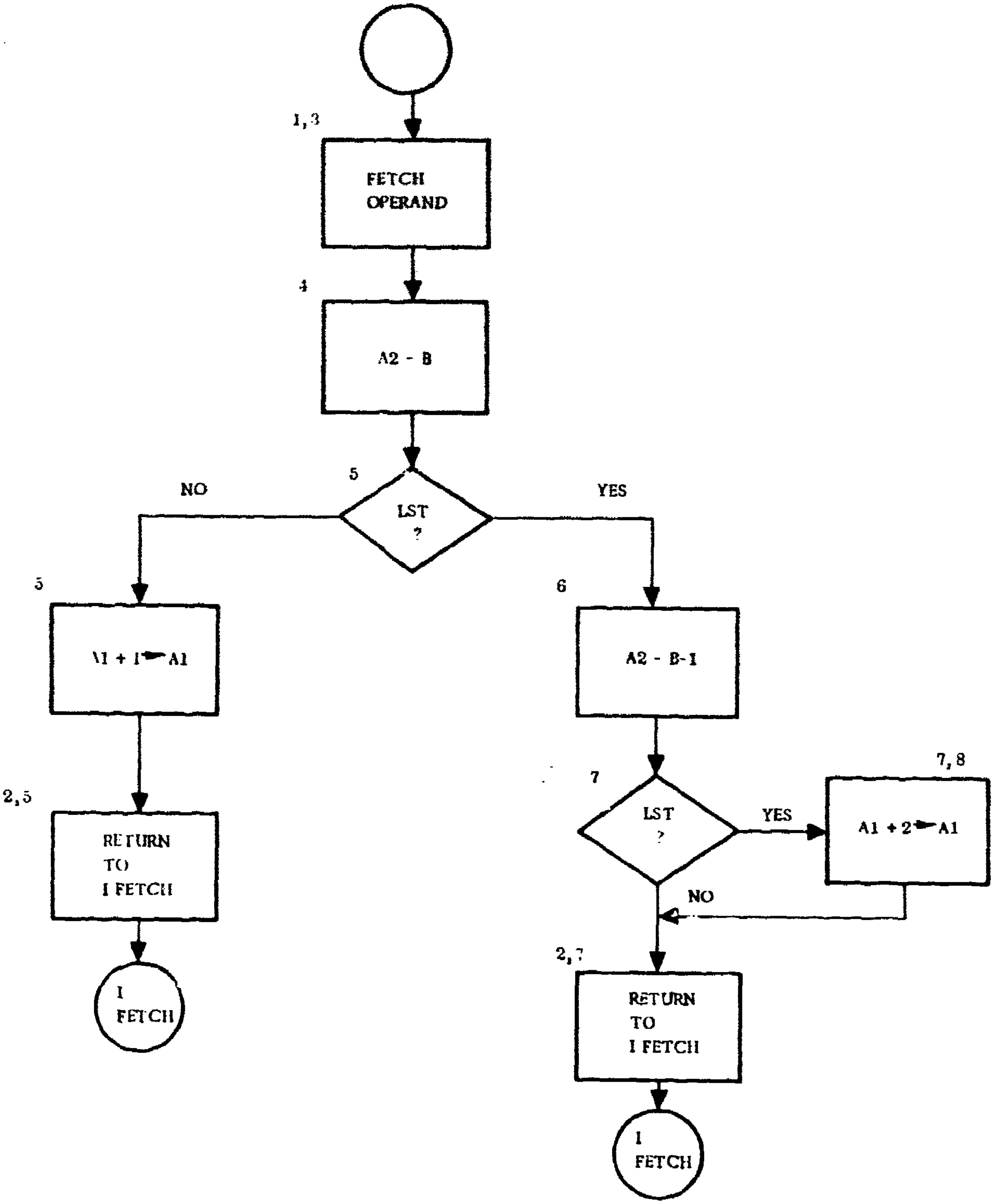


Figure 4-20. Compare Full Word Instruction Execution
(1/2 Word Instruction Format)

Table 4-10. Microcode for 1/2 Word Load Execution

Microinstr	Microcode	Comments
1	Ext Op uncond, MR2	
2	Type 2, Load SAR with 16	
3	Type 2, Load AMPCR	With jump address to I fetch
4	LU cond, SC = RDC, SC = 0 then wait, BEX	
5	LU cond, SC = LC1, Left Shift B, BSW → A2, SC = 1 then jump otherwise step	
6	LU uncond, Right Shift B, BSW → B	
7	LU uncond, Left Shift B, BSW → A2, Successor jump	

Table 4-11. Microcode for Compare Full Word Execution

Microinstr	Microcode	Comments
1	Ext Op uncond, MR2	
2	Type 2, Load AMPCR	With jump address to I fetch
3	LU cond, SC = RDC, SC = 0 then wait, BEX	
4	LU uncond, A2 → A, BTTT → B, SUB A&B	
5	LU cond, $\overline{SC}$ = LST, SC = 1 then step, SC = 0 then jump, A1 → A, B100 → B, Add A&B, BSW → A1	
6	LU uncond, A2 → A, BTTT → B, SUB A& (B-1)	
7	LU cond, SC = LST, SC = 0 then jump, A1 → A, B100 → B, Add A & B, BSW → A1	
8	LU uncond, A1 → A, B 100 → B, Add A&B, BSW → A1, Successor = Jump	

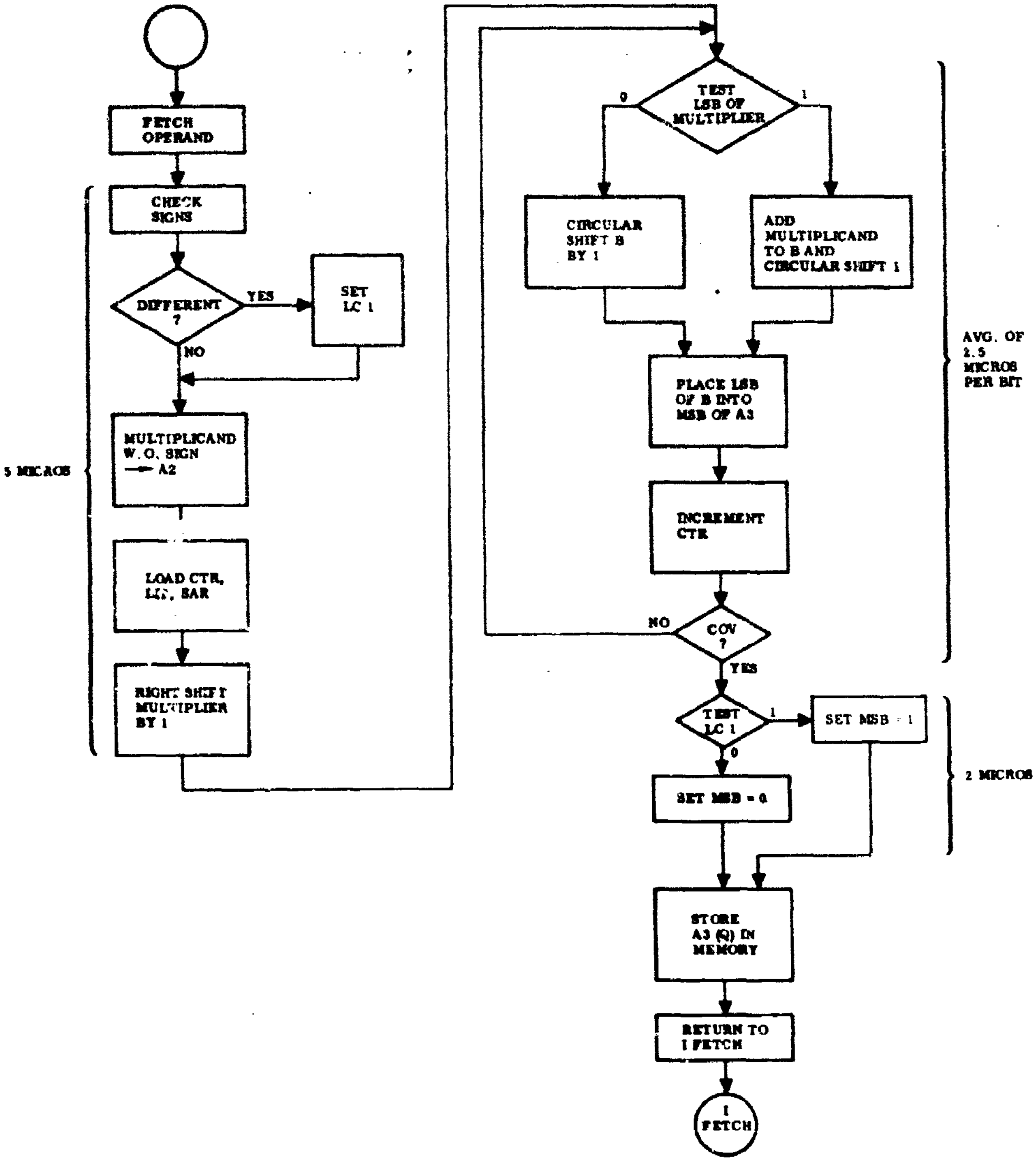


Figure 4-21. Multiply Full Word Execution

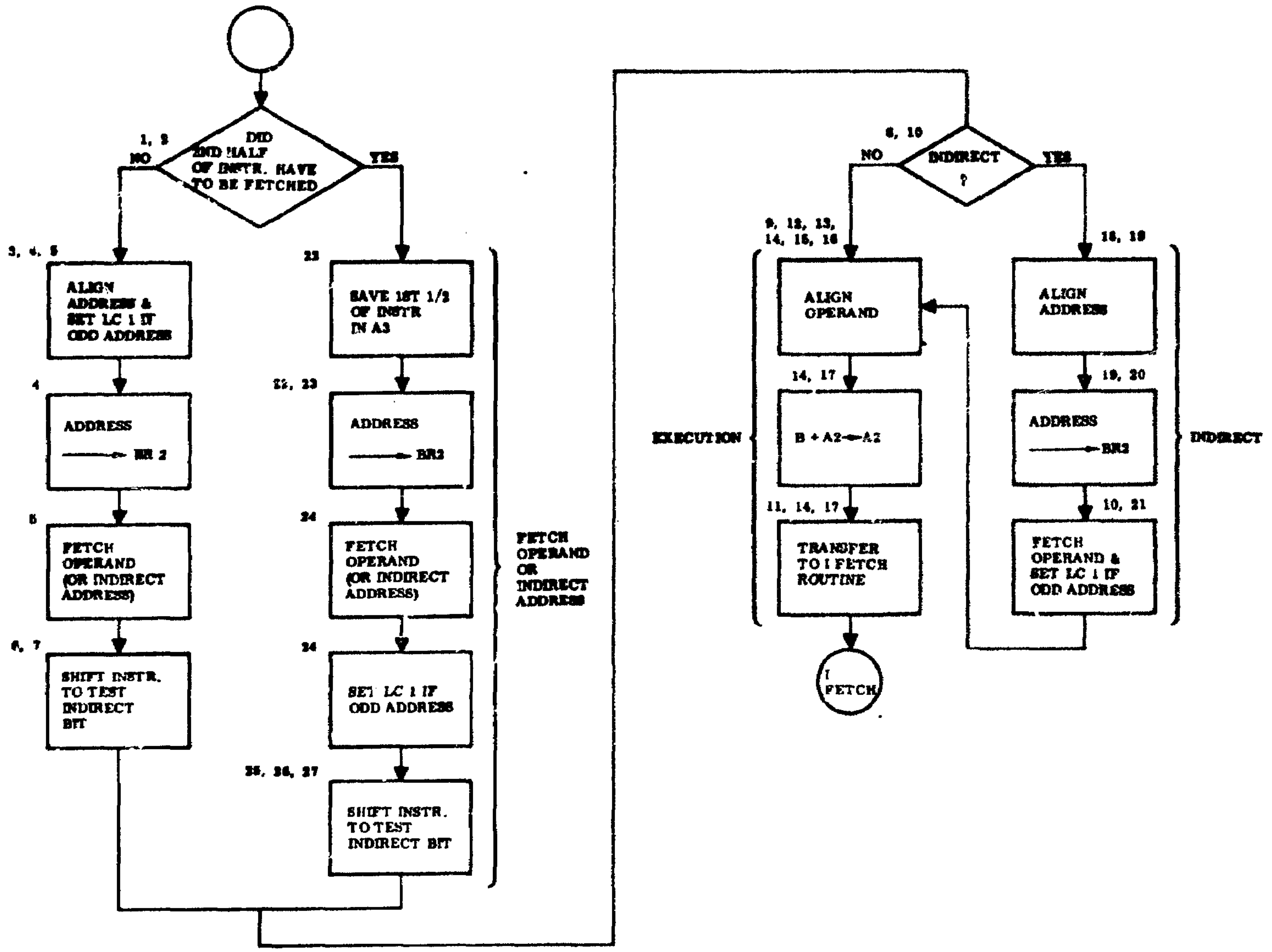

DID 2ND HALF OF INSTR. HAVE TO BE FETCHED
1, 2
NO
YES
3, 4, 5
ALIGN ADDRESS & SET LC 1 IF ODD ADDRESS
4
ADDRESS ——→ BR 2
5
FETCH OPERAND (OR INDIRECT ADDRESS)
6, 7
SHIFT INSTR. TO TEST INDIRECT BIT
23
SAVE 1ST 1/2 OF INSTR IN A3
22, 23
ADDRESS ——→ BR2
24
FETCH OPERAND (OR INDIRECT ADDRESS)
24
SET LC 1 IF ODD ADDRESS
25, 26, 27
SHIFT INSTR. TO TEST INDIRECT BIT
FETCH OPERAND OR INDIRECT ADDRESS
8, 10
INDIRECT ?
NO
YES
9, 12, 13, 14, 15, 16
ALIGN OPERAND
14, 17
B + A2 → A2
11, 14, 17
TRANSFER TO I FETCH ROUTINE
I FETCH
EXECUTION
18, 19
ALIGN ADDRESS
19, 20
ADDRESS ——→ BR2
10, 21
FETCH OPERAND & SET LC 1 IF ODD ADDRESS
INDIRECT

Figure 4-22. Half Word Add Instruction Format Interpretation and Execution (Full Word Instruction Format)

Table 4-12. Microcode for 1/2 Word Add Instruction Format Interpretation and Execution

Microinstr	Microcode	Comments
1	Type 2, Load AMPCR	With jump address to 22
2	LU uncond, SC = LC1, SC = 1 then jump otherwise step	
3	Type 2, Load SAR with 22	
4	LU uncond, Right Shift B, BSW → BR2	
5	Ext Op uncond, MR2, SC = MST, SC = 1 then set LC1	
6	Type 2 Load SAR with 8	
7	LU uncond, Right Shift B	
8	Type 2, Load AMPCR	With jump address to indirect loop, 18
9	Type 2, Load SAR with 16	
10	LU Cond, SC = MST, SC = 1 then Call otherwise step	
11	Type 2, Load AMPCR	With jump address to I fetch
12	LU Cond, SC = RDC, SC = 0 then wait,BEX	
13	LU Cond, SC = LC1, Left Shift B, BSW → B SC = 1 then Skip otherwise Step	
14	LU uncond, A2 → A, BTTT → B, Add A&B, BSW → A2, JUMP	
15	LU uncond, Right Shift B, BSW → B	
16	LU uncond, Left Shift B, BSW → B	
17	LU uncond, A2 → A, BTTT → B, Add A&B, BSW → B, Successor = Jump	

Table 4-12. (Cont)

Microinstr	Microcode	Comments
18	LU cond, SC = RDC, SC = 0 then wait, BEX	
19	LU cond, SC = LC1, Left Shift B, BSW → BR2, SC = 1 Skip otherwise Step	
20	LU uncond, 0 → A, BTTT → B, Add A&B, BSW → BR2	
21	Ext Op uncond, MR2, SC = MST, SC = 1 then set LC1, Successor = Jump	
22	LU cond, SC = RDC, SC = 0 then wait, 0 → A BTTT → B, Add A&B, BSW → A3, BEX	
23	LU uncond, 0 → A, BTTT → B, Add A&B, BSW → BR2	
24	Ext Op uncond, MR2, SC = MST, SC = 1 then set LC1	
25	Type 2 Load AMPCR	With jump address to 8
26	Type 2 Load SAR with 2	
27	LU uncond, Left Shift A3, Successor = Jump	

The flow charts given above enable the total instruction execution times to be calculated in interpreter clocks and time for memory responses. It was assumed that the interpreter clock was 0.25 μsec (4 MHz) as in the present design. The memory response time was calculated based on the SWI timing analysis presented in Section 4.3.1. The memory cycle times assumed were 0.5 μsec read cycle and 0.8 μsec write cycle which is compatible with the 2-mil plated wire memory technology to be used in the packaging characteristics portion of this study. The total memory read time including SWI delays is therefore 1.0 μsec. This assumes no added delays due to priority resolution in a multiprocessor configuration. The degradation factors due to multiprocessor configurations will be considered separately.

The timing results for the short format instructions are summarized in Table 4-13. It is seen that the times are variable and depend on factors such as whether addressing is relative to the program counter (the A1 register) or to one of the index registers (stored in memory), whether the operand address is even or odd, etc. A listing of typical short format instruction times is given in Table 4-14. It is estimated that the average short (add, load, store, etc) type of instruction will take 11 μsec and the multiply will take 32.75 μsec. These are complete times and include, I fetch, I interpretation, operand fetch, and I execution.

Table 4-13. Instruction Execution Times in μsec for Short Format Instructions

Function	Addressing Relative to	
	P Ctr	Index Reg
I FETCH	3.75	3.75
I Format Interpretation	4.75	5.75
Subtotal	8.50	9.50
I EXECUTION		
1. Full Word add, Figure 4-18	1.75	1.75
2. 1/2 Word load, Figure 4-19		
a. Even address	2.25	2.25
b. Odd address	1.75	1.75
3. Full Word compare, Figure 4-20	2.00-2.75	2.00-2.75
4. Multiply, Figure 4-21 average	23.75	23.75

Table 4-14. Typical Execution Times for 1/2 Word Format Instructions

Instruction	Time μsec
Add	10.25 - 11.25
Add 1/2	10.50 - 12
Compare	10.50 - 12.25
Compare 1/2	11.00 - 13.00
Load	10.25 - 11.25
Load 1/2	10.25 - 11.75
Store	10.25 - 11.25
Store 1/2	10.25 - 11.75
Sub	10.25 - 11.25
Sub 1/2	10.50 - 12.00
AND	10.25 - 11.25
OR	10.25 - 11.25
EXCL OR	10.25 - 11.25
Multiply	32.25 - 33.25
Average Short Instr	11.00 μsec
Average Multiply	32.75 μsec

The 1/2 word add instruction using a full word instruction format (Figure 4-22) took the following times:

1. No indirect, odd operand and instruction address — 7.75 μsec
2. No indirect, even operand and instruction address — 8.75 μsec
3. Indirect, even operand and instruction address — 10.50 μsec

The complete instruction time, as seen, ranged from 7.75 to 10.50 μsec. A similar range is expected for other short type of instructions by extrapolating from the experience in Table 4-14. Based on experience in the FB-111A/F-111D avionics system it is expected that very few instructions will use the indirect addressing format (3 percent from statistics on the referenced system). Therefore the following average times are expected for instructions using the full word (32 bit) instruction format:

1. Short (add, load, store, etc) — 8.75 μsec
2. Multiply — 30 μsec

At this point, an estimate of the interpreter's throughput capability can be made, when operating in an emulation mode for the IBM 4π CP avionics computer. An instruction mix representing various types of operations derived from the FB-111A/ F-111D avionics system will be used to derive the throughput in operations per second.

As stated in Section 2 the percentage of long and short instructions expected is 30 percent and 70 percent respectively. This leads to the following average instruction times:

$$\text{time}_{\text{Short}} \text{ (add, store, load, etc)} = 0.7\ (11) + 0.3\ (8.75) = 10.3\ \mu\text{sec}$$

$$\text{time}_{\text{MPY}} = 0.7\ (32.75) + 0.3\ (30) = 32\ \mu\text{sec}$$

The relative frequency of occurrence of the various types of operations is given in Table 2-3 and shows 89 percent are short type (add, load, store, logical, branch, etc) and 11 percent are long type (MPY, divide). This leads to the following throughput in operations per second

$$t_{\text{avg}} = 10.3 \times 0.89 + 32 \times 0.11 = 12.68\ \mu\text{sec}$$

$$\text{throughput} = 10^6/12.68 = 79{,}000 \text{ operations/second}$$

The other aspect of the interpreter that needs to be considered in the emulation mode of operation is the size of the MPM and NM. The following approximations were made to arrive at these sizes.

1. MPM

   a. I fetch – 23 micros + 1 x 65 op codes = 88
   b. Short Instr format Interpretation – 31 micros + 1 x 17 op codes = 48
   c. Short Instr execution – 7 micros (avg) x 17 op codes = 119
   d. Short Instr execution (not using I format interpretation) – 30 micros (avg) x 8 op codes = 240
   e. Long Instr format interpretation and execution – 25 micros (avg) x 29 op codes = 725

   Total = 1220

2. NM

   A rough estimate indicated that approximately 784 of these are Type 1 microinstructions. If it is assumed that half of these are common, then:

   Total = 392

4.4.3 Emulation Optimized with Macros

It is obvious, when considering the above data in Section 4.2, that there is a significant overhead in the I fetch and I format interpretation of every instruction.

This is one of the reasons for considering macros that optimize the interpreter to the application. Macros can perform more complex operations in a single instruction thereby eliminating a significant overhead that goes along with every instruction in Section 4.2.

Two types of macros were considered to optimize the emulation presented in Section 4.2. Data was available on each thereby offering an estimate of the quantitative improvement from the use of macros. The first type of macro was placing conventional subroutines such as sine, arc tan, etc in MPM and using a single macroinstruction to execute each. The other type of macro was specialized complex instructions that could replace sequences of code.

The first type of macro considered, conventional subroutines, was based on statistics from the FB-111A/F-111D avionics system. The data from this system showed that approximately 44 percent of execution time was spent in subroutines. Further most of this time was due to ten subroutines:

1. Sine Cosine
2. Square Root
3. Bin dec/Bin dec half
4. Arc tan/Arc tan half
5. Euler, Euler C_k
6. Limit
7. $P_{costart}$
8. Root Sum Square
9. Synchro
10. Matrix 3 x 3, Matrix 3 x 3 T

Some of these subroutines were examined to determine the throughput improvement by placing them as macros in the MPM. It was found that in most cases a throughput improvement factor of two resulted, i.e. it took half as long to execute the subroutine. For example, in the conventional subroutine manner, the subroutine for sine and cosine would have taken ~580 μsec; however as a macro it took 261 μsec. It was also noted that the principal limiting factor in this improvement was the speed of multiply. In the 261 μsec for the sine macro, 85 percent of this time was spent in multiply. The same points hold true for most of the other subroutines converted into macros. Therefore, if the subroutines are converted into macros, the speed requirements of the avionics system can be reduced by 22 percent (1/2 of 44 percent). It should also be noted that this does not have any significant impact on the main storage requirements.

The other type of macro considered was converted sequences of conventional instructions into a single macro. A recent study conducted by Autonetics (Ref 14) considered the use of macros for an advanced tactical missile inertial guidance system. Extensive statistics were gathered in this study and many are applicable to the avionics application under consideration here.

In the referenced study many macros were considered. In addition to the conventional subroutines already considered above, the following macros along with the percent of time spent in them were noted:

Macro	% Execution Time Used by
VXSC − Vector x Scalar	11%
VAD3 − Vector Add (1 x 3)	5.5%
VSU3 − Vector Sub (1 x 3)	6%
VXFR − Transfer Vector in memory	1%
Matrix Multiply 1 x 3 x 3 x 1	8%
Matrix Multiply 2 x 3 x 3 x 1	7%
	38-1/2%

In the vector add and vector subtract a significant improvement should be realized if they are placed in MPM. The others will be limited by the speed of multiply, however, even here there will be at least a 1/3 improvement since the I fetch and interpretation are eliminated. Therefore a factor of two is again a good approximation for the throughput improvement due to these types of macros. This will give a 19 percent reduction in the speed requirements.

Considering the combined effects of the two types of macros; the first type has effected 44 percent of the required throughput, there then remains 56 percent that can be improved with the second type of macro. If this 56 percent can be reduced by 19 percent, this gives a net reduction of 11 percent for the second type of macro. The net effect for the two types of macros can be summarized as follows:

Type of Macro	Throughput Requirements Reduction
Subroutine	22%
Complex instruction	11%
Net Effect	33%

The net effect of macros can then be expressed in either of two ways:

1. The throughput requirements are reduced by 33 percent, or

2. The throughput capability of an interpreter is increased by 50 percent and the requirements remain the same.

The second type of macro, complex instructions, also has an effect on the main memory storage requirements. Many instructions are now replaced by a single instruction. It should be noted that the first type of macro, subroutines, has no effect on the main memory requirements. Therefore, since 38-1/2 percent of the operations per second are effected by the complex instruction macros, an estimate of the instructions effected by these macros can be made. If it is assumed that these effected operations per second actually represent 1/2 of the total instructions (allowing for the fact that many will be in high iteration rate loops), then 19.25 percent of the instructions are converted into complex macros. It is estimated, considering the complexity of the macros, that each complex macro will replace approximately five instructions giving an effective reduction of 80 percent to the 19.25 percent of the total instructions. This gives a net reduction of 15.5 percent to the total instruction requirements.

In summary the use of macros has the following effect on the computational requirements:

1. Speed – requirements reduced by 33 percent

2. Storage – instructions reduced by 15.5 percent

The other factor that needs to be considered in the emulation mode optimized with macros, is the additional MPM and NM required. The subroutines and complex macros were examined and it was estimated that they would add approximately 647 words to the MPM and 324 words to the NM. This brings the total requirements to:

1. MPM – 1867 words

2. NM – 716 words

4.4.4 Multiprocessor Characteristics and Capabilities

In the previous sections the capabilities of the interpreter have been defined for two of the possible modes of operation identified in Section 4.1:

1. Emulation:

   Speed – 79,000 operations/sec

2. Emulation optimized with macros for typical avionics application:

   Speed – requirements reduced by 33 percent or define capability as 119,000 operations/sec

   Storage – requirements for instruction storage reduced by 15.5 percent

These capabilities are based upon emulation of a conventional state of the art avionics computer such as the IBM 4π CP.

The capabilities of the Burroughs multiprocessor as defined above hold true for a single interpreter. More interpreters may be added, thereby increasing the throughput capability of the multiprocessor. however due to conflicts of interpreters accessing memory in multiprocessor configurations, the increase in throughput is not simply the sum of the individual interpreters throughput which could be defined as an ideal multiprocessor. A multiprocessor degradation factor can be defined as one minus the effective throughput divided by the ideal multiprocessor througput (percent degradation compared to ideal case). The degradation factor can be expected to increase as more interpreters are added. This factor is very difficult to specify. It depends on a number of parameters such as the type of executive utilized, the effort put into (and the resultant cost of) programming for a multiprocessor environment, the characteristics of the problem being run, and the particular data the problem is being used on.

Little information is available on the degradation factors in multiprocessing systems and any information available is very application dependent. Therefore for lack of any better information the following table of degradation factors will be assumed as shown in Table 4-15.

Table 4-15. Multiprocessor Capability

Number of Interpreters	Degradation Factor	Relative Throughput	Total Throughput (ops/sec)
1	0	1.00	79,000/119,000
2	0.10	1.80	142,500/213,750
3	0.20	2.40	190,000/285,000
4	0.25	3.00	237,000/355,500
5	0.30	3.50	277,000/415,500

Another consideration, in defining multiprocessor configurations, is the relative number of interpreters to memory modules. In general, the number of memory modules should be equal to or greater than the number of interpreter modules otherwise conflicts in access to memory will increase and the degradation factor will increase.

5. CONFIGURATION DEFINITION

5.1 CENTRAL VS LOCAL PROCESSING ALLOCATION

5.1.1 Introduction

The objective of this portion of the study was to specify the recommended allocation of the processing tasks. The tasks were considered either allocated to a central processor or to local processors associated with the subsystems. The processing requirements were defined in Section 2 in summary form, i.e., the requirements for each major function were given. The requirements used in the processing allocation were at a lower level; the requirements of individual tasks within a major function were used. These task requirements are contained in detail in Appendix A. Further, in using these task requirements a number of assumptions had to be made regarding overhead functions such as executive. input/output. synchronization, test, etc. The following estimates were made for these overhead factors:

	Local Processing	Central Processing
SPEED:	Add to Basic Requirements*: 7 percent for executive, 15 percent for sync communication/test, 50 percent spare	Add to Basic Requirements*: 20 percent for executive. 50 percent spare
STORAGE:	Add to Basic Requirements : 30 percent for a short/long instruction format, 7 percent for executive, 15 percent for sync/communication/test, 50 percent spare	Add to Basic Requirements**: 30 percent for a short/long instruction format, 15 percent for executive, 50 percent spare

*: Operations/sec represent a mix of instruction types as specified in Table 2-3.
**: Assuming requirements are independent of word length as explained in Section 2.2.

It should be noted that these overhead factors are only approximations at this point in time and are included to arrive at a realistic sizing of the local and central processors. The 50 percent spare is a requirement as specified in Section 2.2.

The sizing of the local and central processors will be performed using the capabilities defined in Section 4.4. The mode of operation will be assumed to be emulation optimized by macros. Table 4-15 defines the throughput capability for a multiprocessor with up to five interpreters. The throughput capability in operations per second represents a mix of instruction types as defined in Table 2-3. As noted in Section 4.4, with this mode of operation, the storage requirements for instructions should be reduced by 15.5 percent.

5.1.2 Candidate Allocations

The processing requirements specified in Section 2 were defined based upon central processing. These requirements were analyzed to determine which could be performed in local processors. The processing requirements consist of ten major functions. Five functions were found to be amenable to local processing; namely, Navigation, Steering, Target/Checkpoint Acquisition, Weapon Delivery, and Penetration Aids. The remaining five were not suitable for local processing for a variety of reasons. Terrain Following/Avoidance and Mission and Traffic Control are growth functions and are presently not included in the digital processing requirements. Mission Data Management and Central Integrated Test are functions that involve processing of data for all other major functions and all the subsystems; as a result, these functions cannot be efficiently allocated to local processors. The Executive function is an overhead function that is required to one degree or another in all the processors.

Each of the five functions is comprised of various tasks as defined in Appendix A. The functions were analyzed to determine tasks within each function that could be allocated to local processing. Some of the factors considered in arriving at task allocations were:

1. The I/O rate and number of I/O signals of a particular task with other tasks within the same function, with other functions, and with the hardware items associated with each subsystem.

2. The prerequisite tasks required to be executed before the allocated task is executed.

3. The number and type of hardware items interfaced with the allocated task.

Due to the large number of tasks within each function, it is not possible to examine all possible combinations of tasks within a function allocated to local processors. Based upon the factors noted above, each function was analyzed to define reasonable candidates for allocation to local processors. The results are presented in Tables 5-1 through 5-10, and Figures 5-1 through 5-5. For each function there is a table that defines the candidate allocations considered for analysis, a table that presents the processing requirements for each candidate, and a figure that graphically presents the processing requirements (the processing requirements, speed and storage, are for the local processor).

As an example, Table 5-1 defines eight candidate allocations for local processing for the Navigation function. The numbers 1.1, 1.4 etc., in this table refer to the numbering system adapted in the requirements analysis description in Appendix A; 1.1 is the task IMU Control-Fast, 1.4 is the task IMU Control-Filter, etc.

Table 5-2 defines the local processor speed and storage, the I/O rate and number of signals between the subsystem (including the local processor, if any) and the central processor, and any pertinent considerations in a remarks column. The speed is specified in operations/second, and represents a mix of instructions (add, multiply, etc.) as specified in Table 2-3, and as used in deriving the throughput capability of an interpreter in Section 4.4. The storage is specified in number of 16 bit words. In

Table 5-1. Navigation Processing Allocation

Number	CANDIDATES:	Tasks in Local Processor
1.		None
2.	1.1	IMU Control - Fast
3.	1.2	IMU Control - Mid
4.	1.1, 1.2, 1.3, 1.4	All IMU Subtasks
5.	1.1, 1.3, 1.4	IMU Control - Fast, - Slow, - Filter
6.	1.1 through 1.12	All IMU, Grd. Align., and Nav Subtasks except Nav-Filter
7.	1.1 through 1.13	All IMU, Grd. Align., and Nav Subtasks
8.		All

this part of the processing allocation analysis, the 15.5 percent reduction in number of instructions required is not taken into account, overall, this typically is 11 percent since the storage is typically represented by 73 percent instructions and 27 percent data. The I/O rate specifies the number of 16 bit words/sec transmitted between the central processor and the subsystem (including any local processors). Figure 5-1 simply contains a graphical representation of Table 5-2.

5.1.3 Recommended Allocation

In general, the ultimate criteria upon which allocation decisions are based are factors such as cost, reliability, and physical parameters such as size and weight. This assumes that whatever is being traded off meets performance requirements or goals. There are many items which directly affect the above factors. The items which are considered pertinent to this allocation analysis are listed below:

1. The data rates on the I/O data bus

2. The management and technical interface between subsystems and the central processor

3. The impact of design changes

4. Subsystem reliability

5. The processing load

Table 5-2. Navigation Function Allocation Requirements

Candidate Allocation	Local Processor Speed (ops/sec)	Local Processor Storage (Words)	I/O Rate Subsystem * -Central Processor (words/sec)	I/O Signals Subsystem-Central Processor	Remarks
1	-	-	3,104	235	
2	33,337	1,036	2,928	263	Only interfaces with IMU's
3	69,652	2,079	2,548	270	1.2 is a prerequisite to 1.5
4	104,913	3,669	2,270	196	1.2 prerequisite to 1.5; 1.4 prerequisite to 1.7
5	35,261	2,502	2,870	193	Only interfaces with IMU's
6	231,897	12,010	3,388	516	1.7 prerequisite to 1.15
7	243,100	30,337	3,266	362	
8	243,100	32,312	2,866	246	

*Includes any local processors

Table 5-3. Steering Processing Allocation

CANDIDATES:

Number		Subtasks in Local Processor
1.		None
2.	2.1, 2.6	Lateral Steering - Fast
3.	2.1, 2.2, 2.5, 2.6, 2.7, 2.10	All of Lateral Steering
4.	2.3, 2.4, 2.8, 2.9	All of Pitch Steering
5.		All

Table 5-4. Steering Function Allocation Requirements

Candidate Allocation	Local Processor Speed (ops/sec)	Local Processor Storage (words)	I/O Rate Subsystem-Central Processor (words/sec)	I/O Signals Subsystem-Central Processor	Remarks
1	-	-	448	40	
2	2,200	800	644	66	
3	24,420	5,120	862	142	1.11 is prerequisite to this allocation
4	3,700	2,864	580	50	2.1 and 2.2 are prerequisites to this allocation
5	28,120	7,472	794	152	

Table 5-5. Target/Checkpoint Acquisition Processing Allocation

CANDIDATES:

Number		Subtasks in Local Processor
1.		None
2.	3.1, 3.2, 3.6	Cursor Control, FLR Control, Altitude Calib.
3.	3.2, 3.6	FLR Control, Altitude Calib.
4.	3.3	EVS Control
5.		All

Table 5-6. Target/Checkpoint Acquisition Allocation Requirements

Candidate Allocation	Local Processor Speed (ops/sec)	Local Processor Storage (words)	I/O Rate Subsystem-Central Processor (words/sec)	I/O Signals Subsystem-Central Processor	Remarks
1	-	-	1142	43	
2	44,622	1,964	2356	130	1.8 and 3.5 are prerequisites to this allocation, 3.2 is a prerequisite to 3.3, and 3.6 is a prerequisite to 3.7, only interfaces with FLR
3	42,254	1,340	2260	75	3.1 and 3.5 are prerequisites to this allocation
4	30,784	908	1634	70	3.2 is prerequisite to this allocation
5	107,004	4,207	2974	202	1.8, 1.9, and 1.11 are prerequisites to this allocation

Table 5-7. Weapon Delivery Processing Allocation

CANDIDATES:

Number		Subtasks in Local Processor
1.		None
2.	4.1, 4.2, 4.3, 4.4	Bomb Release Level Deliv. - Fast and Slow Drogue Deliv.
3.	4.1, 4.3, 4.4	Bomb Release, Level Deliv. - Slow, Drogue Deliv.
4.	4.7, 4.8, 4.9	All of SRAM Delivery
5.		All

Table 5-8. Weapon Delivery Allocation Requirements

Candidate Allocation	Local Processor Speed (ops/sec)	Local Processor Storage (words)	I/O Rate Subsystem-Central Processor (words/sec)	I/O Signals Subsystem-Central Processor	Remarks
1	-	-	5952	1715	
2	84,841	6,464	6926	1802	3.1 is a prerequisite to this allocation, only interfaces with SLU
3	46,361	5,024	6453	1768	4.2 is a prerequisite to this allocation
4	43,475	11,012	412	100	1.8, 1.12, 5.11 are prerequisites to this allocation
5	153,994	25,100	945	85	

Table 5-9. Penetration Aids Processing Allocation

CANDIDATES:

Number		Subtasks in Local Processor
1.		None
2.	5.1	Identify IR Threat
3.	5.1, 5.2	Identify IR Threat, IR Track File Proc.
4.	5.3	RF Known Emitter Sort
5.	5.3, 5.4	RF Emitter Sort, RF Characteristics
6.	5.3 through 5.7, 5.14	All RF Processing
7.	5.9, 5.10, 5.11	All CM Processing
8.	5.12, 5.13	All TSD Command Proc.
9.	5.3 through 5.7, 5.12 through 5.14	All RF Proc. and All TSD Command Proc.
10.		All

Table 5-10. Penetration Aids Allocation Requirements

Candidate Allocation	Local Processor Speed (ops/sec)	Local Processor Storage (words)	I/O Rate Subsystem-Central Processor (words/sec)	I/O Signals Subsystem-Central Processor	Remarks
1	-	-	16,320	969	
2	93,980	4,832	19.048	1086	1.8 is prerequisite to this allocation
3	127,132	8,672	18.068	1058	
4	37,888	9,152	6,912	388	1.9 is prerequisite to this allocation
5	52,096	11,072	6,144	340	
6	175,734	36,492	10,622	678	
7	37,888	6,752	16,224	991	5.8 is prerequisite to this allocation
8	19,536	6,272	16,488	1004	5.11 and 1.12 are prerequisites to this allocation
9	195,270	42,292	8,646	537	1.9, 1.12, and 5.11 are prerequisites to this allocation
10	407,666	61,637	7.519	293	

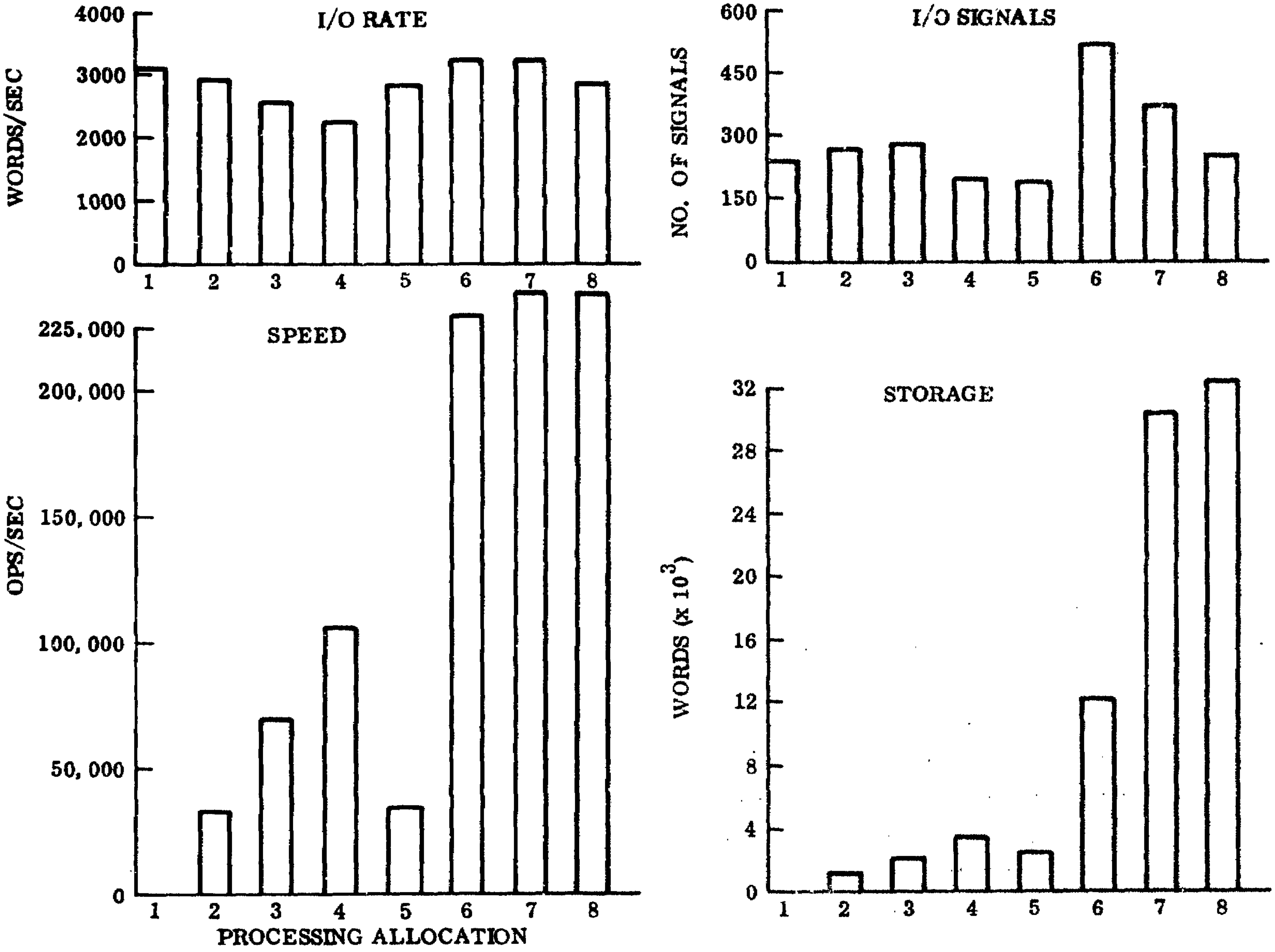


Figure 5-1. Function 1 - Navigation

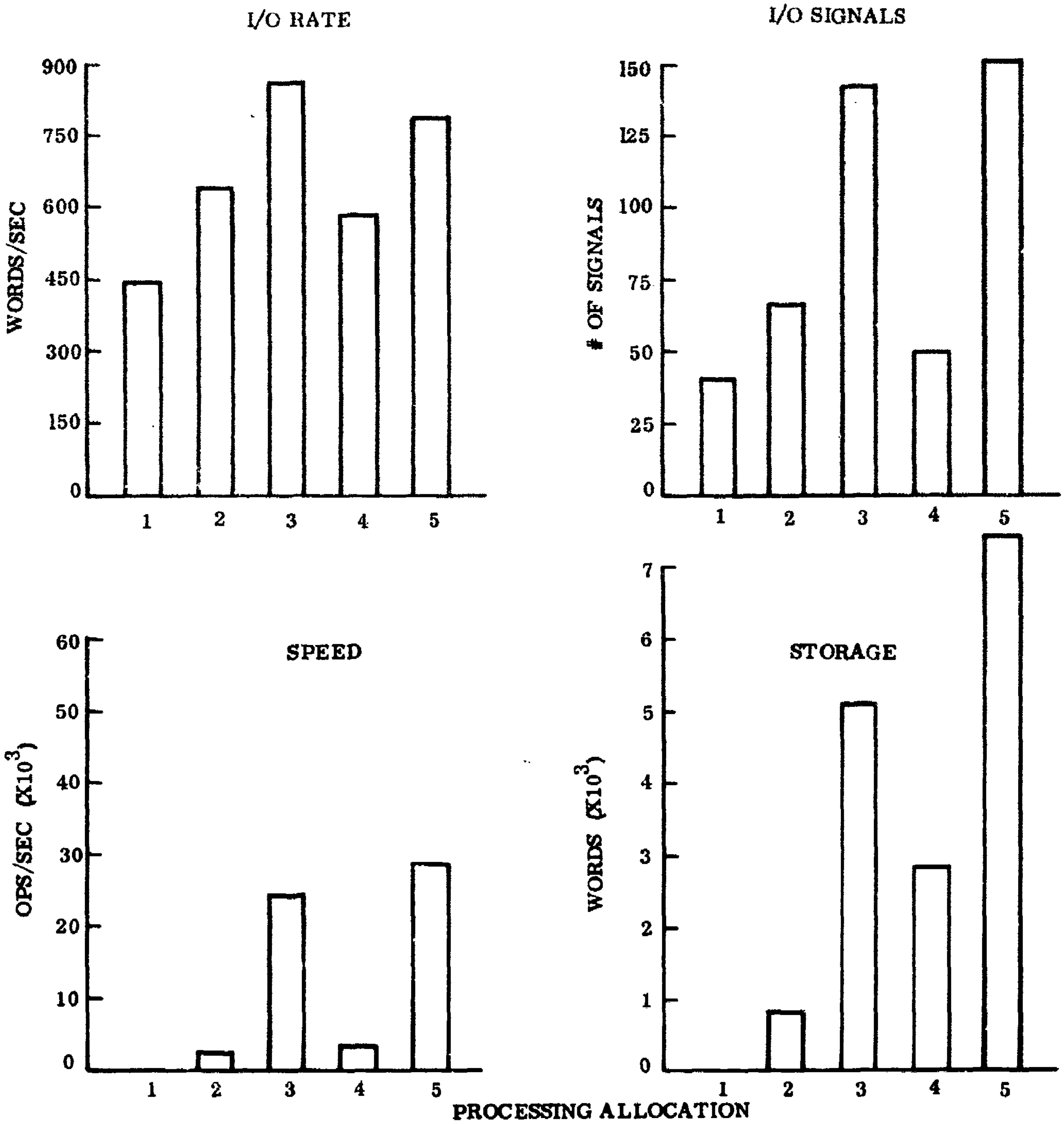


Figure 5-2. Function 2 - Steering

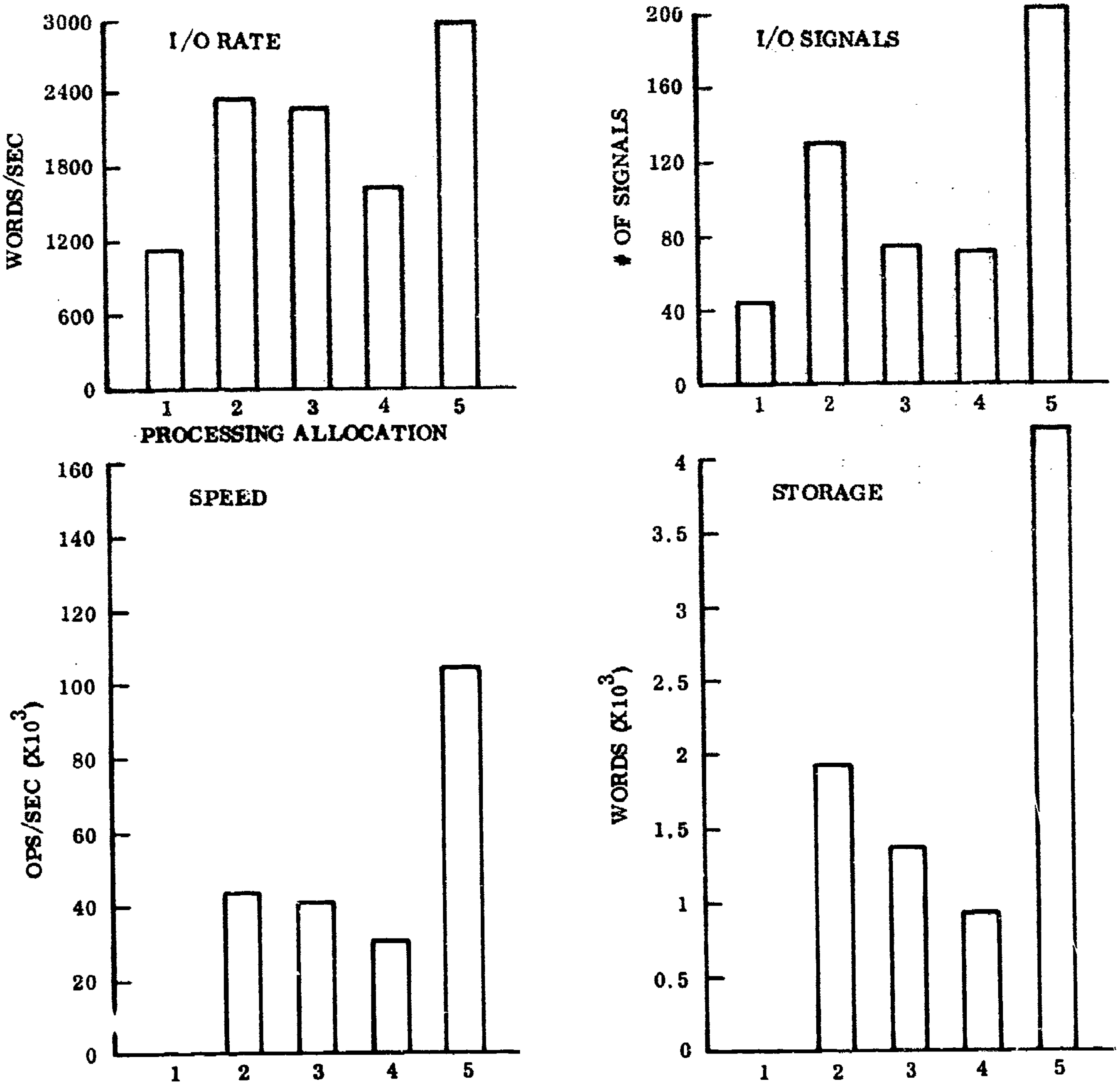


Figure 5-3. Function 3 - Target/Checkpoint Acquisition

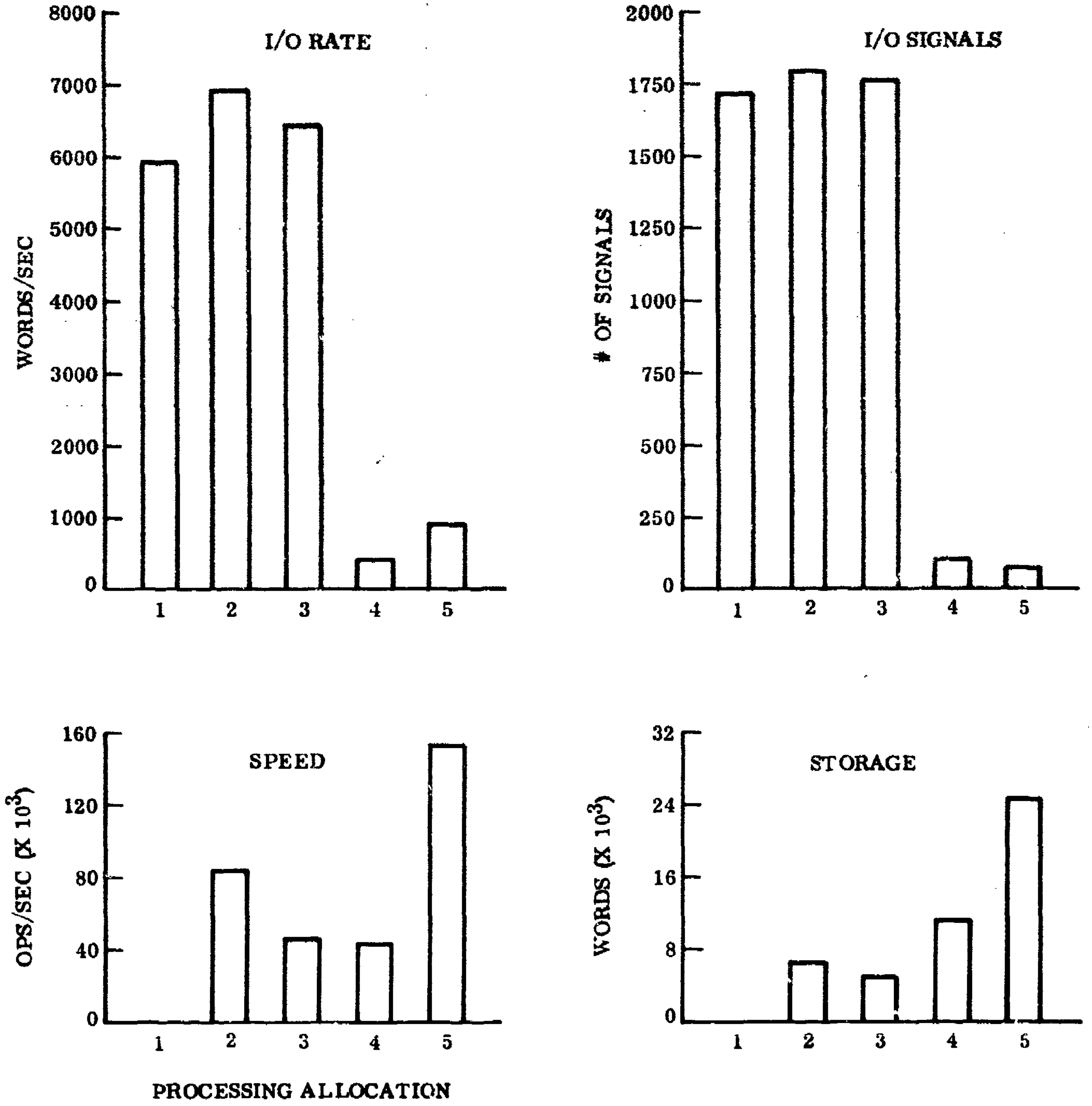


Figure 5-4. Function 4 - Weapon Delivery

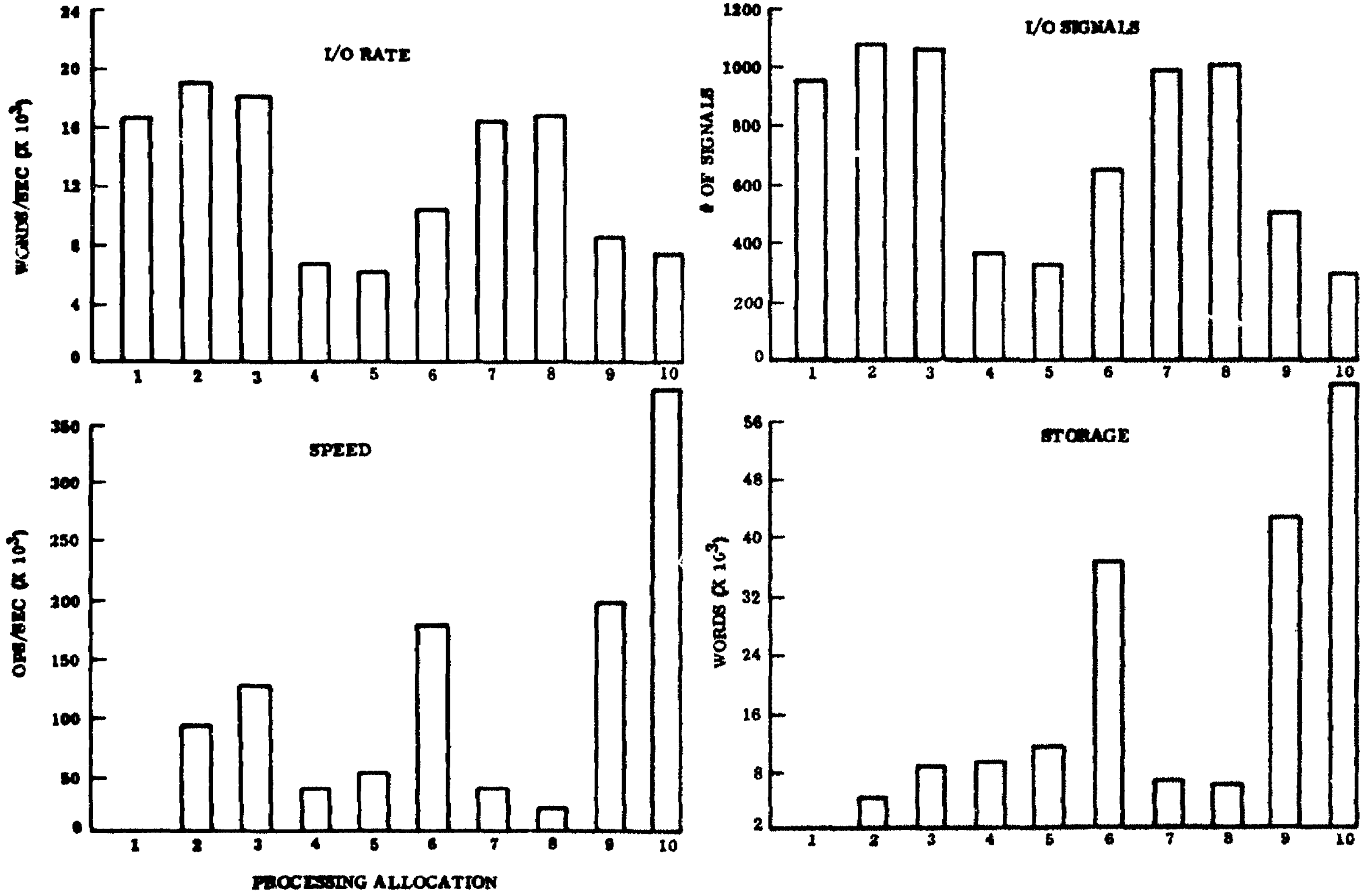


Figure 5-5. Function 5 - Penetration Aids

In a system design study these items may be quantitatively related through suitable mathematical expressions to system design decision criteria, e.g., cost and reliability. Such an effort must encompass the entire system which in this case would be the entire air vehicle. This type of effort is beyond the scope of this study. The justification and rationale for considering the above items as quantitative measures for the central versus local processing allocation in this study is given below.

Item 1 is measured directly by the I/O data rate between the subsystem (including any local processors) and the central processor. This item affects the complexity of the information transfer system and hence its cost. On the other hand, if an information transfer system exists with a certain capability, then a reduction in data rate with a fixed capability provides increased spare, growth, and provision for more redundancy and error checking.

Item 2 is difficult to measure. It involves the complexity in implementing the interface between a subsystem and the central processor. This complexity can be measured by the number of signals, or items of information required to be transferred, since each signal requires documentation, testing, validation, design interface between different manufacturers, etc. However, it should be noted that this is definitely not an absolute measure since many other factors, regarding the type and nature of the signals, also need to be considered. Some of these factors are any critical timing or synchronization required in the interface, e.g., signals may be required at precise intervals of time, signals may require the precise synchronization of other events to reception of these signals, etc. These type of factors are difficult to quantify at this point for the allocation analysis. Nevertheless, they must be kept in mind when comparing strictly the number of signals in an interface.

Item 3 is a difficult factor to quantify. However, it should be apparent that a design change, that only affects a local processor, which has a simpler program than the central processor, will have a lower cost impact than if the local processing functions were in the central processor with the resultant design change affecting the central processor program. This results from the fact that design changes require increased effort as the size and complexity of the program changed increases and that program validation and checkout increases with the size and complexity of the program changed. Likewise hardware changes to a local processor should be simpler to handle than to the central processor. Therefore, in general, the cost of design changes will be reduced by utilizing local processing rather than central processing.

Item 4, subsystem reliability, can be increased in certain cases by utilizing local processing. This may be true for subsystems that can provide autonomous functions through a local processor in the event of central processor failure. In such cases the reliability of the local processor and not the central processor enters into the successful performance of such functions. The reliability of the local processor in these cases may be required to be greater than the central processor. A typical example of this situation is the provision of local processing with an IMU to provide autonomous navigation capability in the event of central processor failure (such as the F-111 avionics system).

Item 5, the processing load, on the central processor determines the complexity, feasibility and technical risk involved in meeting the requirements. If the processing load is excessive, the use of local processing may reduce the load on the central processor, reducing its complexity and in some cases making it feasible at a reasonable technical risk. In this system, the speed capability of the interpreter is a limiting factor in meeting the system requirements, storage requirements do not appear as a limiting factor on the interpreter. Therefore, an important consideration is the use of local processing to reduce the speed requirements on the central processor.

In summary, the following items are desired to be accomplished in performing the local vs central processing tradeoff:

1. Reduce the data rates on the I/O data bus.

2. Reduce the management and technical interfaces between subsystems.

3. Reduce the cost of design changes through minimization of impact on the system of such changes.

4. Increase system reliability.

5. Reduce the central processor load to minimize the central processor complexity.

Quantitative data on Items 1, 2 and 5 was prepared for each of the candidate processing allocations. Items 3 and 4 are difficult to measure at this point in time. These factors may be measured when the ASB system design is specified in more detail.

The relative importance of all the tradeoff factors is what ultimately leads one to a decision on allocation; this is extremely difficult to specify quantitatively. It is normally accomplished after cost and technology parameters are specified for the complete system. Thus far in this study, with the knowledge of the processing requirements and the preliminary estimate of the interpreter's capability, it is apparent that Item 5 is very important if not critical to the successful implementation of the system. It is therefore highly desirable to place as much of the processing speed load in local processors as possible. It is also desirable to perform this while reducing the complexity of the interface between the subsystem and the central processor.

Table 5-2 and Figure 5-1 gives the quantitative data on the navigation function candidate allocations. Examination of this data and evaluating the relative interface reduction and speed reduction shows that three candidates should be examined more closely, Candidates 4, 5, and 8. The following observations are noted regarding the interface:

1. Candidate 4 - All IMU tasks done locally

   a. Reduces I/O signals from 235 to 196 (almost minimum)
   b. Reduces I/O rate from 3,104 to 2,276 (minimum)
   c. No prerequisites to these tasks, they are prerequisites only to tasks in ground align mode
   d. Interfaces with the two IMU's and the two FDC's

2. Candidate 5 - All IMU tasks except IMU Control - Mid done locally

   a. Reduces I/O signals from 235 to 193 (minimum)
   b. Reduces I/O rate slightly, 3,104 to 2,870
   c. Same prerequisites as in Candidate 4
   d. Only interface is with the two IMU's

3. Candidate 8 - Entire navigation function done locally

   a. Slight increase in I/O signals from 235 to 246
   b. Slight decrease in I/O rate from 3,104 to 2,866
   c. Interfaces with large number of hardware items
   d. Many other functions have as prerequisites tasks within the navigation function; strict synchronization and timing of the central processor to the local processor may be required

The following observations are noted regarding the processing load:

1. Candidate 4

   a. Takes approximately 105,000 ops/sec off the central processor, would require one interpreter
   b. Low storage reduction, approximately 3.5 K

2. Candidate 5

   a. Low speed reduction, approximately 35,000 ops/sec
   b. Low storage reduction, approximately 2.5 K

3. Candidate 8

   a. Significant speed reduction, approximately 250,000 ops/sec
   b. Significant storage reduction, approximately 32,000

Candidate 4 is the recommended allocation (all IMU tasks done locally). It reduces the I/O signals almost to the minimum, minimizes the I/O rate, and takes a considerable speed load off the central processor. Perhaps most important, but difficult to quantify, is the simpler interface with this allocation. High rate closed loop control computations for the IMU are performed locally. This enables a simpler selloff and validation of performance when the subsystem contains the IMU control computations. This allocation will require one interpreter and one 4K memory module*. Candidate 8 would only be recommended if a further speed reduction in the central processor is required and this consideration outweighs the increase in interface complexity.

Table 5-4 and Figure 5-2 show the data on the candidates for the steering function and Table 5-6 and Figure 5-3 show the data on the candidates for the Target/Checkpoint Acquisition function. Neither of these two functions appear to have suitable candidates

*The memory modules are specified in 32-bit word length. The requirements in the corresponding tables are equivalent 16-bit words.

for local processing. The I/O rate, I/O signals, and interface complexity are increased for all candidates over central processing. The steering function offers very little speed reduction. The Target/Checkpoint Acquisition function offers a considerable speed reduction (approximately 107,000 ops/sec) if all of it is done locally, however, this complicates the interface considerably (4 x number of I/O signals, 2 x I/O rate). In addition, many of the tasks in both functions require the navigation function as prerequisites. The recommended allocation is central processing for these two functions.

Table 5-8 and Figure 5-4 show the data on the candidates for the Weapon Delivery function. Examination of the five candidates and evaluating their relative interface complexity and speed reduction indicates that two should be examined closely, namely 4 and 5. The following observations are noted on these candidates:

1. Candidate 4 - All of SRAM delivery tasks

   a. Significant reduction in I/O rate from 5,952 to 412 (minimal)
   b. Significant reduction in I/O signals from 1,715 to 100 (almost minimal)
   c. Some speed reduction on central processor, approximately 43,000 ops/sec
   d. Some storage reduction, approximately 11,000
   e. Some Navigation and Penetration Aids functions are prerequisite to these tasks

2. Candidate 5 - All of Weapon Delivery Function

   a. Significant reduction in I/O rate from 5,952 to 945
   b. Significant reduction in I/O signals from 1,715 to 85 (minimal)
   c. Considerable speed reduction, approximately 154,000 ops/sec
   d. Some Navigation, Penetration Aids, and Target/Checkpoint Acquisition functions are prerequisites

The choice between these two candidates is difficult to make. Both offer similar reductions in I/O rate and number of I/O signals. However, the management interface is simpler with Candidate 4, all SRAM done locally, since it does not interface directly with other functions and has a relatively simple interface with other tasks in the weapon delivery function. Candidate 5 does not appear to be an overly complex interface either if done locally, however it is not as straightforward as Candidate 4. On the other hand, Candidate 5 offers a considerable speed reduction compared to Candidate 4. Therefore, Candidate 5 would be recommended due to the importance of speed reduction, however, if, upon further examination of the central processor complexity, Candidate 5 can be reasonably placed in the central processor, then Candidate 4 would be the preferred choice. Candidate 4 will require one interpreter and one 8K memory module and Candidate 5 will require two interpreters and two 8K memory modules.

Table 5-10 and Figure 5-5 show the data on the candidates for the Penetration Aids function. Examination of the ten candidates indicates that Candidate 10 is the best choice. It minimizes the I/O interface in terms of I/O signals, I/O rate (nearly minimal), and management interface complexity and also offers a significant reduction in speed on the central processor. However, this application requires a multiprocessor with five interpreters, the maximum allowable. It should be noted at this point that it is desirable, in this allocation analysis, to use at the most four interpreters. This allows one spare interpreter to be placed in a multiprocessor if desired. Therefore, alternate allocations using sets of multiprocessors in a multicomputer mode were also considered for this function.

Four sub allocations were considered:

1. Sub Allocation 1

   IR tasks (5.1 and 5.2) in one multiprocessor and remaining Penetration Aids tasks in another multiprocessor

2. Sub Allocation 2

   RF tasks (5.3 through 5.7, 5.14) in one multiprocessor and remaining Penetration Aids tasks in another multiprocessor

3. Sub Allocation 3

   IR and TSD (Threat Situation Display) tasks (5.1, 5.2, 5.12, 5.13) in one multiprocessor and remaining tasks, RF, CM (countermeasures), and threat correlation in another multiprocessor.

4. Sub Allocation 4

   IR tasks (5.1, 5.2) in one multiprocessor. RF tasks (5.3 through 5.7, 5.14) in another multiprocessor, and TSD, CM, and threat correlation (5.8 through 5.13) in another multiprocessor.

The processing requirements and multicomputer configurations required for each sub allocation are:

1. Sub Allocation 1

MP (multiprocessor) 1:	134,000 ops/sec 9,100 words Two interpreters Two 4K memory modules

MP2:	295,000 ops/sec 56,100 words Four Interpreters Four 8K memory modules

2. Sub Allocation 2

MP1: 185,000 ops/sec
38,400 words
Two Interpreters
Three 8K memory modules

MP2: 244,000 ops/sec
26,900 words
Three Interpreters
Three 4K memory modules

3. Sub Allocation 3

MP1: 154,000 ops/sec
15,200 words
Two Interpreters
Two 4K memory modules

MP2: 274,000 ops/sec
49,300 words
Three Interpreters
Three 8K memory modules

4. MP1: 134,000 ops/sec
9,100 words
Two Interpreters
Two 4K memory modules

MP2: 185,000 ops/sec
38,400 words
Two Interpreters
3 8K memory modules

MP3: 104,724 ops/sec
18,400 words
One Interpreter
One 8K memory module

All of these sub allocations take a similar amount of hardware. Sub allocation 1 requires the most interpreters, six compared to five in all the other sub allocations. Sub allocation three minimizes the amount of hardware used; however, it results in a potentially complex management interface between the two multiprocessors. In this sub allocation one multiprocessor does the IR processing and the other the RF processing. In addition, each multiprocessor does some of the processing that requires both IR and RF processing results. If one multiprocessor were to be programmed by one manufacturer (e.g., the RF subsystem supplier) and the other by a different manufacturer (e.g., the IR subsystem supplier), the management interface between these two suppliers would increase in this sub allocation compared to the other sub allocations.

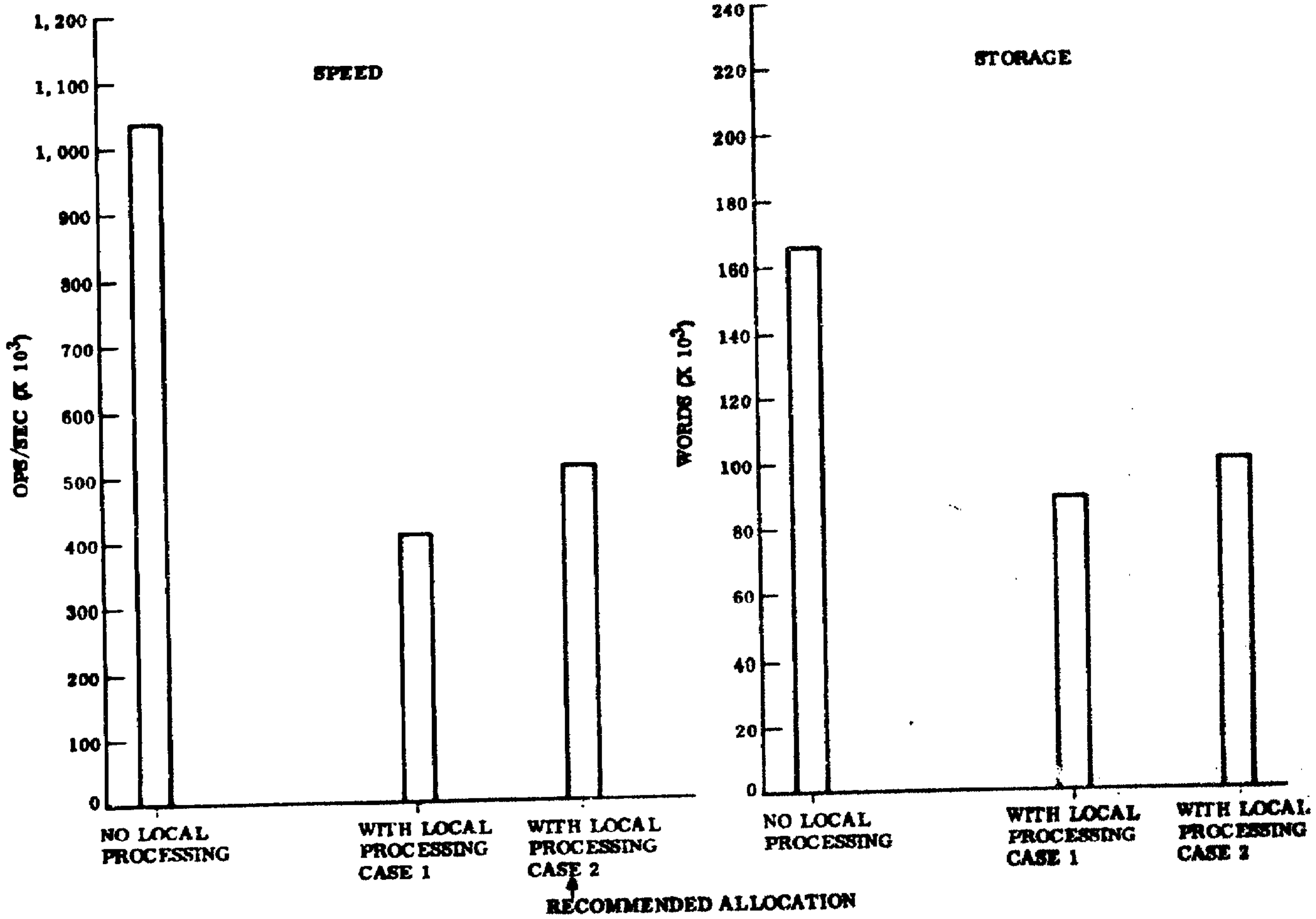
Comparing sub allocations 2 and 4, the latter requires slightly more hardware (same number of interpreters, 4K more memory, three versus two SWI) however it results in a much simpler interface between the multiprocessors. One multiprocessor is used for IR processing (MP1), another for RF processing (MP2), and the other for functions that use outputs of both of these multiprocessors (MP3). MP1 and MP2 do not require any intercommunication, they simply interface with MP3. MP1 and MP2 take the form of true local processors and may be considered part of the IR and RF subsystems, respectively. In this situation the IR and RF subsystem may be more easily sold off, tested and validated by their respective suppliers. In addition, the reliability and redundancy requirements for MP1 and MP2 may be chosen by the subsystem supplier to meet the requirements specified for each particular subsystem. Therefore, it is felt that the slight increase in hardware complexity for sub allocation 4 will be offset by the simpler interface requirements and is the recommended approach for the Penetration Aids function.

At this point, specific local processing configurations have been developed for the Navigation, Weapon Delivery, and Penetration Aids functions with the remaining functions performed in the central processor. The requirements remaining on the central processor will now be examined. These requirements are reduced as shown in Figure 5-6 (case 1) to the following:

Function		Speed (ops/sec)	Storage (words) *
1.0	Navigation	124,800	26,300
2.0	Steering	27,400	6,500
3.0	Target/Checkpoint Acquisition	104,000	4,050
7.0	Mission and Data Management	31,600	9,600
9.0	CITS	120,000	41,500
	Total	407,800	87,950

* 16-bit words, executive overhead distributed over all functions, 50 percent spare capacity included.

The speed requirement requires a five interpreter multiprocessor. As noted above, it is desirable to use only four interpreters functionally, thereby allowing a spare interpreter to be provided in a multiprocessor. In addition, there exist some I/O processing requirements to be added to the above requirements which will probably exceed the capability of a five interpreter multiprocessor. Therefore, either more functions/tasks must be allocated to local processing or a multicomputer configuration must be used for the central processor. The remainder of the Navigation function could be allocated to local processing. This approach appears undesirable since it increases the interface with the subsystem and increases the management interface between the navigation subsystem and the central processor. Many other



NOTE: SPEED REPRESENTS A MIX OF OPERATIONS TYPES PER SECOND, STORAGE IS IN 16 BIT WORDS, OVERHEAD AND 50% SPARE FACTORS ARE INCLUDED

Figure 5-6. Central Processing Requirements

functions have as prerequisites the tasks in the Navigation function. If these tasks were performed locally, the communication and synchronization between functions would become considerably complex if not unmanageable.

The functions in the central processor were examined to determine if a multicomputer configuration could be readily implemented. The CITS function is most suitable of all the functions to be placed in one multiprocessor, due to interface complexity considerations with the remaining functions in another multiprocessor. The multicomputer configuration then takes the form of two multiprocessors with the following requirements:

1. Multiprocessor 1 - CITS
   126,000 ops/sec*
   41,500 words

2. Multiprocessor 2 - Navigation, Steering, Target/Checkpoint Acquisition, Mission Data Management
   302,000 ops/sec
   46,500 words

*Additional overhead is added to operate as a multicomputer.

Multiprocessor 1 requires two interpreters and three 8K memory modules (recall these are 32-bit word modules) and Multiprocessor 2 requires four interpreters and five 4K memory modules.

This is a reasonable candidate to consider for the central processor. Two additional candidates were examined to determine if the processing load and functions could be more evenly split between the two multiprocessors. The following two sub allocations appear as suitable candidates for equalizing the computational requirements while maintaining a reasonable management interface between the two multiprocessors:

1. Sub Allocation 1

   MP1: CITS, Navigation
   Three Interpreters
   Four 8K memory modules

   MP2: Steering, Target/Checkpoint Acquisition, Mission Data Management
   Two Interpreters
   Two 8K memory modules

2. Sub Allocation 2

   MP1: CITS, Mission Data Management
   Two Interpreters
   Four 8K memory modules

   MP2: Navigation, Steering, Target/Checkpoint Acquisition
   Three Interpreters
   Three 8K memory modules

Either one of these two sub allocations increase the management interface between the two multiprocessors but it is difficult to determine if one is significantly more complex than the other. Therefore, on the basis that it requires slightly less hardware (one memory module), sub allocation 1 is the preferred approach.

At this point a system has been configured for the local processors and the central processor. It is now necessary to examine the central processor configuration to determine if additional capability may be easily added to it. If this is the case, then some of the allocations made previously, that were primarily influenced by speed capability, would have to be re-examined. Multiprocessor, MP2, in the central processor contains two interpreters and may be easily expanded to four interpreters thereby providing additional speed capability.

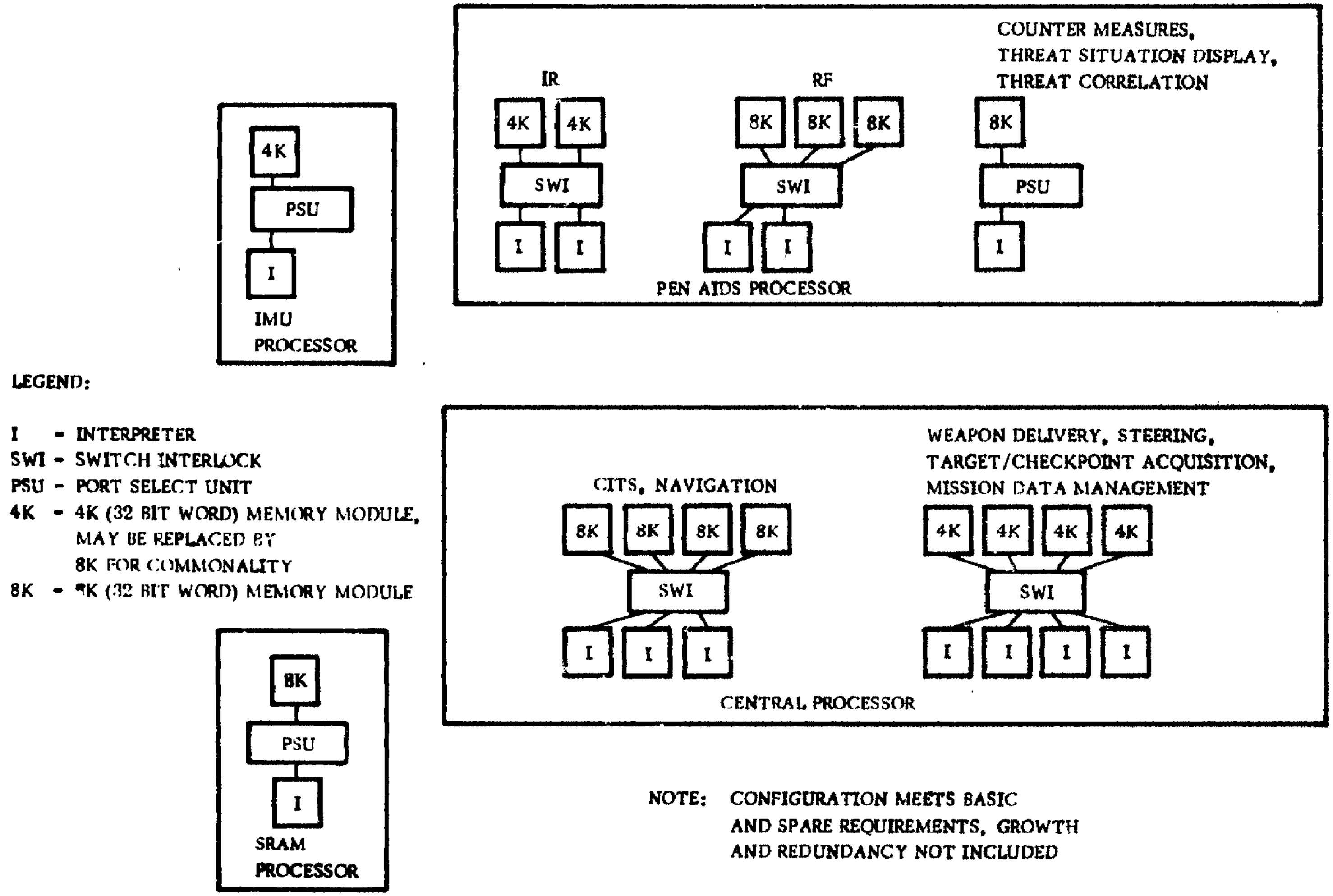
Re-examination of the processing allocations indicates that the allocation selected for the Weapon Delivery function (complete local processing) was based primarily on speed while the other functions were based on both a minimum interface complexity and a speed reduction. Therefore, the approach of using Candidate Allocation Number 4 (all of SRAM delivery done locally) for weapon delivery and thereby bringing into the central computer all the non-SRAM delivery computations should be examined. This allocation (Candidate 4) requires oneinterpreter and one memory module in the weapon delivery subsystem. The resultant configuration in the central processor then takes the following form:

1. Multiprocessor 1 - CITS, Navigation
   256,000 ops/sec
   67,800 words
   Three Interpreters
   Four 8K memory modules

2. Multiprocessor 2 - Weapon Delivery, Steering, Target/
   Checkpoint Acquisition, Mission Data Management
   284,000 ops/sec
   34,000 words
   Four Interpreters
   Four 4K memory modules

This approach has essentially eliminated one interpreter and one memory module at the weapon delivery local processor and added two interpreters and memory modules in the central processor. The interface is considerably simpler with this approach since the weapon delivery computations that interface closely with the navigation, steering, target/checkpoint acquisition, and mission data management functions are now integrated in the central processor. In addition, the local processor in the weapon delivery subsystem now interfaces primarily with the SRAM weapon interface units and not with a variety of equipment in the weapon delivery subsystem. Therefore, this approach has been selected for the weapon delivery processing allocation since it is felt the simplification of the interface will outweigh the additional hardware added to the total system. The overall effects on the central processing requirements of this allocation is shown in Figure 5-6 as local processing - Case 2.

The recommended configuration for the entire system is shown in Figure 5-7 and is summarized in Table 5-11.



LEGEND:

I - INTERPRETER
SWI - SWITCH INTERLOCK
PSU - PORT SELECT UNIT
4K - 4K (32 BIT WORD) MEMORY MODULE,
MAY BE REPLACED BY
8K FOR COMMONALITY
8K - 8K (32 BIT WORD) MEMORY MODULE

NOTE: CONFIGURATION MEETS BASIC
AND SPARE REQUIREMENTS, GROWTH
AND REDUNDANCY NOT INCLUDED

Figure 5-7. Processing Allocation

Table 5-11. Final Configuration for ASB Avionics System

Central Processor:		
Multiprocessor 1	- -	Three Interpreters Four 8K memories
Multiprocessor 2	-	Four Interpreters Four 4K memories
Penetration Aids:		
Multiprocessor 1	-	Two Interpreters Two 4K memories
Multiprocessor 2	-	Two Interpreters Three 8K memories
Multiprocessor 3	-	One Interpreter One 8K memory
Weapon Delivery Processor:		
Single Processor	-	One Interpreter One 8K memory
IMU Processor:		
Single Processor	-	One Interpreter One 4K memory

The memory modules in this configuration consist primarily of 8K modules with some 4K modules. The 4K modules may be replaced with 8K modules, using slightly more hardware, if one desires to maintain commonality among the memory modules.

It should be noted that this configuration is what is required to meet the computational requirements. These requirements include overhead functions and the 50 percent spare factor but exclude the 100 percent growth requirement. The growth can be met in a variety of ways. In some cases additional interpreter and/or memory modules may be added to the appropriate multiprocessor. In other cases a multicomputer configuration may be needed by adding another multiprocessor to meet the growth.

The other factor that needs to be considered in an ultimate configuration is reliability and failure tolerance. Failure tolerance characteristics of the Burroughs Multiprocessor will be investigated in Section 5.3. However, it can be noted that in each of the above processors additional interpreter and memory modules may be provided that would provide failure tolerance. If mass storage can be used to provide backup programs in the event of reconfiguration, then the addition of one interpreter and one memory module to each of the above processors provides the ability to withstand an interpreter or memory module failure with no degradation in performance.

5.2 INTERFACE TO B-1 MULTIPLEX SYSTEM

5.2.1 Introduction

A definition of the B-1 multiplex system is not currently available. A recent study by Radiation, Inc. (Ref 15) presented a study of the B-1 information transfer requirements and the definition of a multiplex system to solve these requirements. This study considered the information transfer requirements of the entire air vehicle, the central computer complex was only a portion of the system. The recommended multiplex system as a result of this study was a multiple bus, baseband, TDM system using a central controller. The central computer constituted one of many terminals connected to a bus and essentially received the same type of service as any other subsystem terminal connected to the bus. It is felt that this system should not be used as the baseline multiplex system for this study for three primary reasons: (1) the approach of treating the central computer as a subsystem and using a central controller may result in considerable problems in timing and synchronization between the computer and subsystems and it is felt this may be an unworkable approach for a real time control central computer system such as the B-1; (2) the status of this multiplex system recommendation is unknown at this time and can only be regarded as a paper design, (3) from what is known of the present B-1 system, the central computer serves as the bus controller.

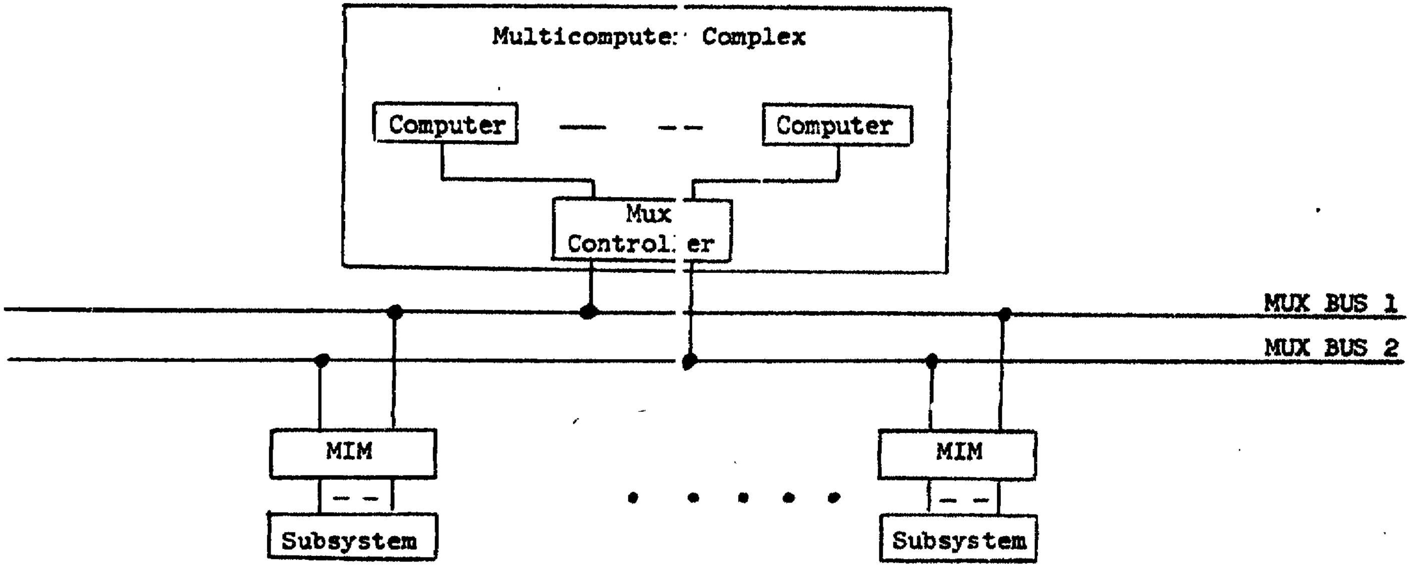
The present B-1 system is configured as shown in Figure 5-8. Two dual redundant buses are used that are driven by a multiplex controller which in turn is controlled by the central computer complex. This system concept will be used as the baseline for this study, however, the present multiplex controller design will not be assumed. The best available description of this multiplex system was found in Ref 16 which is the Rockwell International specification for the MIM, multiplex interface module (Boeing is responsible for the multiplex system specification). Pertinent extracts from the MIM specification are given below in Section 5.2.2 that will provide an understanding of the way the present B-1 multiplexing system will function. It should be kept in mind that in Figure 5-8, the "multicomputer complex" (which is presently in the B-1) corresponds to the central processor derived in this study as shown in Figure 5-7.

5.2.2 Multiplex Interface Module and System Operation*

5.2.2.1 Item Definition

The multiplex interface modules (MIM's) shall provide the interfaces between the B-1 air vehicle multiplex transmission cables and LRU electronics. MIM's shall be configured using two unique units, hereafter referred to as the multiplex interface unit (MIU) and the parallel interface unit(s) (PIU). The MIU and PIU shall be configured so they can be mounted and interconnected on printed circuit cards. All MIU's shall be identical and interchangeable. All PIU's shall be identical and interchangeable. The MIM's shall be capable of receiving/transmitting serial digital data on either of two transmission cables, referred to herein as the multiplex channels or the primary

*This section contains extracts from Ref 16 and does not represent material generated under this study contract.



NOTE: MIM - Multiplex Interface Module

Figure 5-8. Present B-1 Multiplex System

multiplex channel and the secondary multiplex channel. (The above operation shall be under the direction of a controller.) The MIM's shall include transmitters-receivers, coupling transformers, isolation resistors, clock generators, encoders-decoders, self-test logic, signal interface buffers, and the logic necessary to regulate the operation of the MIM multiplex transmission lines and LRU electronics interfaces. The MIM shall be capable of accepting asynchronously data from both multiplex channels, and shall be able to transmit response and data words on the channel in which a valid command word was received.

5.2.2.1.1 Item Diagram. The major components of the MIM and their functional relationship will be as shown in Figure 5-9.

5.2,2.1.2 Interface Definition

5.2.2.1.2.1 Message Format. The message format for the transfer of data to a MIM shall consist of the following forms:

1. One command word, followed by zero to 31 data words, followed by a response word.

2. One command word, followed by no response word (error in the command word received by the MIM).

The message format for data requested from a MIM shall consist of the following forms:

1. One commanded word, followed by a response word, followed by one to 31 data words.

2. One command word, followed by no response word (error in the command word received by the MIM).

The response word shall be generated by the MIM. The command word will be generated by the controller. The data will be generated by the LRU electronics. The MIM will obtain the data from the LRU electronics and format the data into data words and transfer them to the controller.

5.2.2.1.2.2 Word Format. (See Figure 5-10.)

5.2.2.1.2.2.1 Command Word. The command word shall contain the information listed in Table 5-12.

5.2.2.1.2.2.2 Response Word. Except for the sync, the response word shall be an echo of the command word. Therefore, the information contained in the response word shall be identical to the information contained in the command word that caused the MIM to respond. Refer to Tables 5-13 and 5-14. When a command word error is detected. the MIM will not transmit a response word.

5.2.2.1.2.2.3 Data Word. The data word shall contain the information listed in Table 5-15.

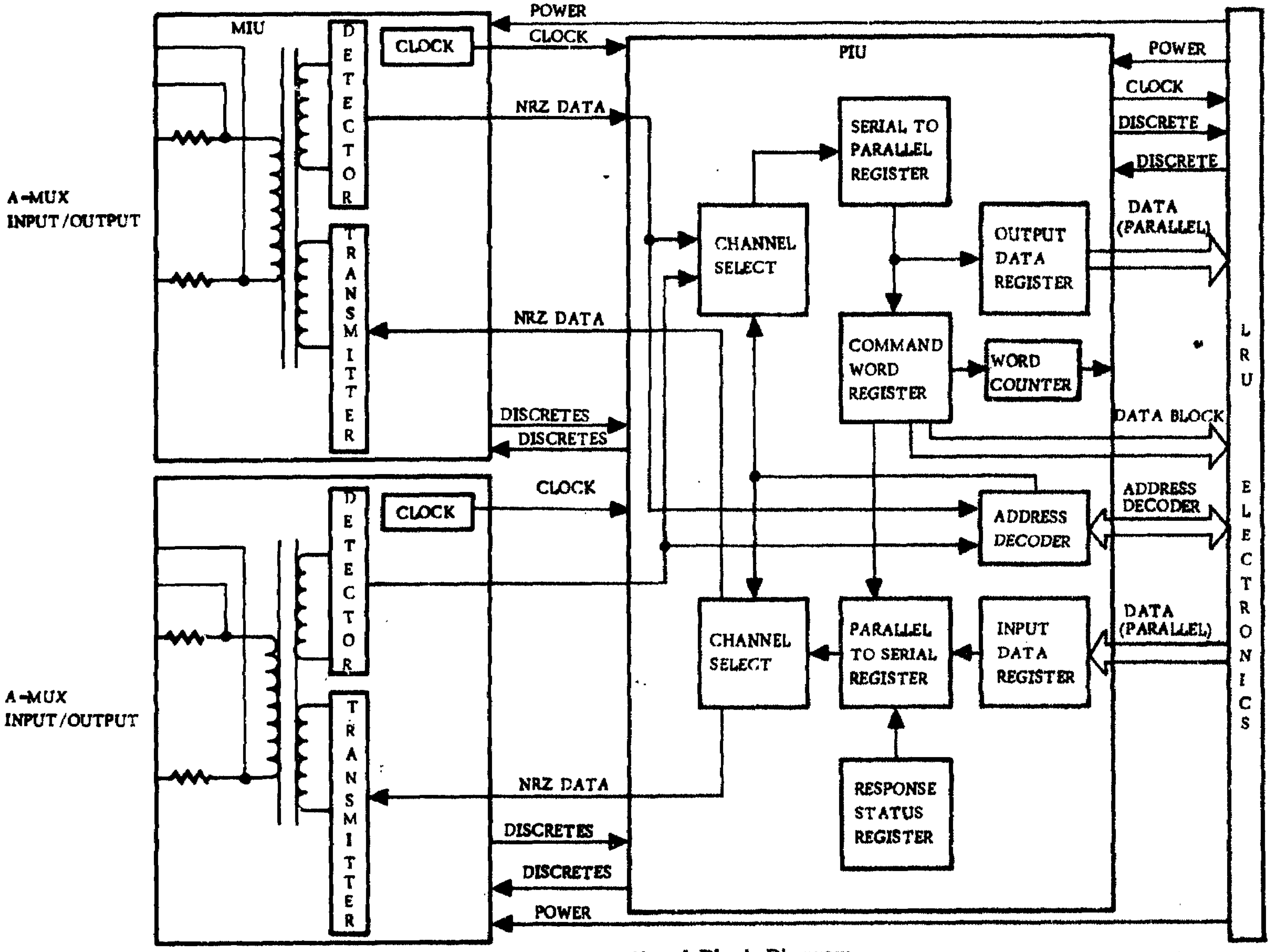


Figure 5-9. MIM Functional Block Diagram

COMMAND WORD FORMAT

1	2	3	4.	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
SYNC			S	V	S	S	MIM ADDRESS					T/R	DATA BLOCK/MODE					NO. OF WORDS					P

RESPONSE WORD FORMAT (VALID DATA TRANSMISSION)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
SYNC			S	V	S	S	MIM ADDRESS					T/R	DATA BLOCK/MODE					NO. OF WORDS					P

RESPONSE WORD FORMAT (INVALID DATA TRANSMISSION)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
SYNC			S	V	S	S	MIM ADDRESS					T/R	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	P

N_i = BIT DENOTING THE TYPE OF ERROR (TBD)

DATA WORD FORMAT

1	2.	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
SYNC			S	V	S	S	DATA													LSB --↵			P

V = VALIDITY DATA S = SPARE

Figure 5-10. Word Format

Table 5-12. Command Word

Item	Bits	Definition
a	1 through 3	Sync bit. Three-bit nonvalid Manchester code.
b	4	Spare (S) bit - unused bit. Unused bits shall be set to logic "0."
c	5	Validity (V) bit. Logic "1": The command word is not usable due to controller's transmitting equipment having a fault condition as determined by self-test and/or monitoring components in the transmitting equipment. Logic "0"" The command word meets acceptability criteria of transmitting unit as determined by its self-test and/or monitoring components.
d	6	Spare (S) bit -unused bit. Unused bit shall be set to logic "0."
e	7	Spare (S) bit - unused bit. Unused bit shall be set to "0."
f	8 through 12	Address bits. A five-bit code that identifies the MIM that shall respond to a given command word. Addresses shall be assigned from 1 to 31. Address C shall be a nonvalid address and a MIM shall not respond to it.
g	13	Transmit/receive (T/R) bit. Logic "1": Commands addressed MIM to transfer requested data. Logic "0": Commands addressed MIM to receive data and activate a mode discrete
h	14 through 18	Data block/mode bits. A five-bit code that identifies the LRU data block starting memory location or commands the LRU into a specific mode/operation.
i	19 through 23	Number of data words. Identifies the number of data words to be transmitted/received. Code 00000 shall be equal to one, code 00001 shall be equal to two, etc, to code 11111 which shall be equal to 32.
j	24	Parity (P) bit. This b shall be set to a value so that the total number of on s in the word is odd.

Table 5-13. Response Word (Valid Data Transmission)

Item	Bits	Definition
a	1 through 3	Sync. Three-bit nonvalid Manchester code.
b	4	Spare. Definition per Table 5-12.
c	5	Validity bit. Definition per Table 5-12.
d	6	Spare. Definition per Table 5-12.
e	7	Spare. Definition per Table 5-12.
f	8 through 12	Address. Definition per Table 5-12.
g	13	Transmit/receive. Definition per Table 5-12.
h	14 through 18	Data block/mode. Definition per Table 5-12.
i	19 through 23	Number of data words. Definition per Table 5-12.
j	24	Parity. Definition per Table 5-12.

Table 5-14. Response Word (Invalid Data Transmission)

Item	Bits	Definition
a	1 through 3	Sync bits. Three-bit nonvalid Manchester code.
b	4	Spare (S) bit - unused bit. Unused bits shall be set to logic "0."
c	5	Validity (V) bit. Same as command word validity bit.
d	6	Spare (S) bits - unused bit. Unused bits shall be set to logic "0."
e	7	Spare (S) bits - unused bit. Unused bits shall be set to logic "0."
f	8 through 12	Address bits. A five-bit code that identifies the MIM which is responding to a command word. Under normal operating conditions (no faults associated with the address logic) the receive command word address and the MIM response word address shall be identical.
g	13	Transmit/receive (T/R) bit. Logic "1": Commands addressed MIM to transfer requested data. Logic "0": Commands addressed MIM to receive data and activate a mode discrete.
h	14 through 23	Bit denoting the type of error (TBD).
i	24	Parity (P) bit. This bit shall be set so that the total number of bits is odd.

Table 5-15. Data Word

Item	Bits	Definition
a	1 through 3	Sync bit. Three-bit nonvalid Manchester code.
b	4	Spare (S) bit - unused bit. Unused bits shall be set to logic "0."
c	5	Validity (V) bit. Logic "1": Data word is not usable due to fault(s) as identified by the LRU's or controller's self-test and/or monitoring components. (The data word shall be transmitted even if the validity bit is a logic "1.") The validity bit on only those word(s) that are affected by a failure(s) shall be set to a logic "1." Logic "0": Data word meets acceptability criteria as determined by the LRU's or controller self-test logic.
d	6	Spare bit - unused bit. Unused bit shall be set to a logic "0."
e	7	Spare bit - unused bit. Unused bit shall be set to a logic "0."
f	8 through 23	Data. Information generated by LRU or controller. Information transmitted in binary, binary-coded decimal (BCD), discrete, or other required forms. Bit 23 is the least significant bit.
g	24	Parity (P) bit. This bit shall be set to a value so that the total number of ones in the word is odd.

5.2.2.1.2.3 MIU/PIU Interface. The MIU/PIU interface shall consist of the signals shown in Figure 5-11. The transfer of data between the MIU and PIU shall be controlled by the MIU. The functions and associated logic levels of the signal shall be as follows:

5.2.2.1.2.3.1 MIU Input-PIU Output Signals.

1. Transmit Enable. The transmit enabie signal in the logic "1" state shall enable the Manchester II encoder portion of the MIU. The encoder shall convert to Manchester II the nonreturn to zero (NRZ) data presented to MIU by the PIU while the transmit enable is a logic "1."

2. Transmit Sync. A Logic "1" pulse 250 to 500 nanoseconds wide shall be provided to synchronize the encoder clocks and output signals. The pulse shall precede each transmission of data to establish the data link dead time between the commands received and the response word(s).

3. Nonreturn to Zero (NRZ) Data In. NRZ data presented to the MIU for transmission on the multiplex channel.

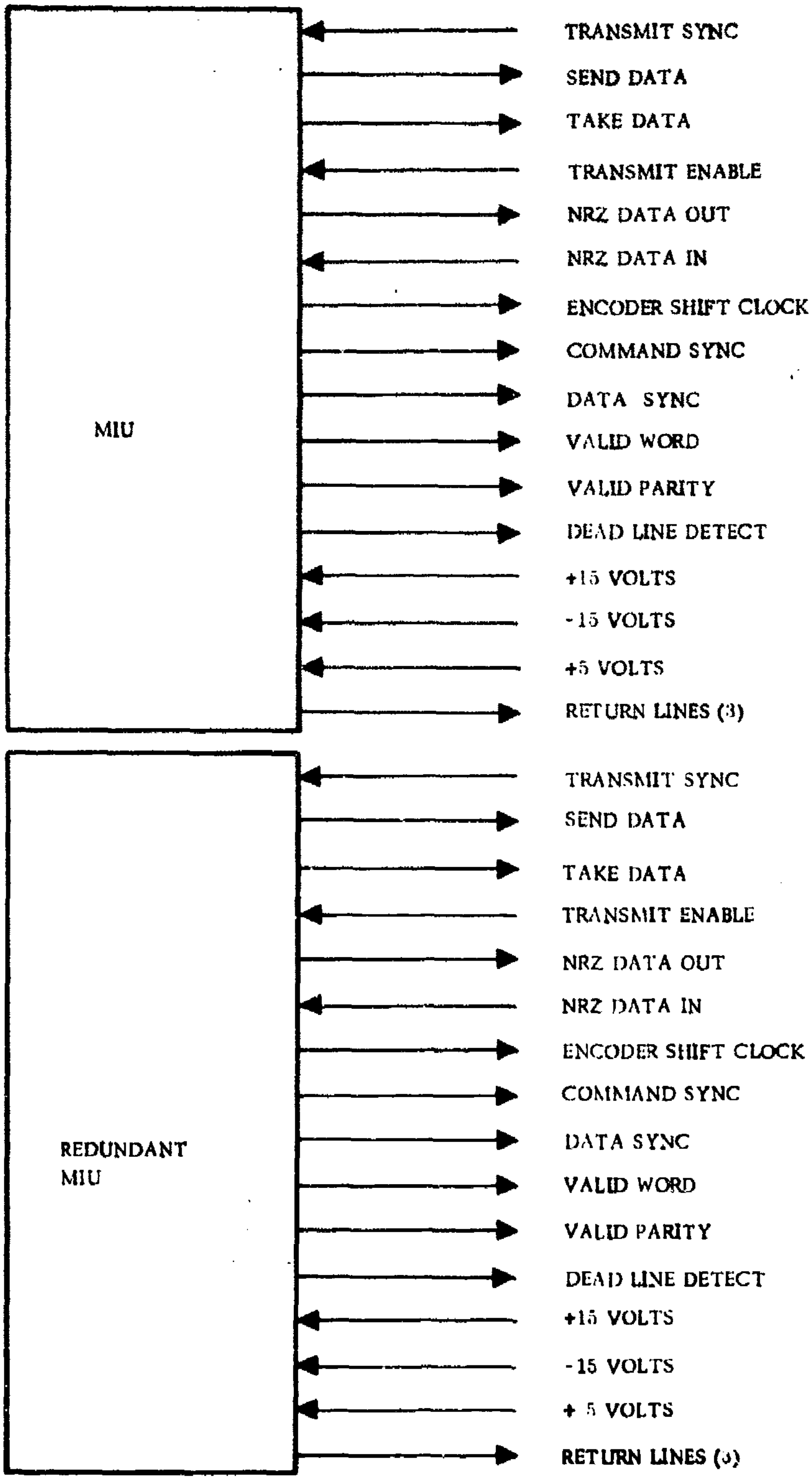


Figure 5-11. MIU/PIU Interface Signals

5.2.2.1.2.3.2 MIU Output-PIU Input Signals

1. Send Data. The send data signal shall be a logic "1" during the period the MIU is capable of accepting and encoding NRZ data in.

2. Encoder Shift Clock. The encoder shift clock shall be provided for synchronizing the transfer of signals.

3. Dead Line Detect. The dead line detect signal in the logic "1" state shall indicate the multiplex channel is activated. In the logic "0" state it will indicate the multiplex channel is inactive.

4. Command Sync. A logic "1" for 21 microseconds following the receipt of a valid command sync.

5. Data Sync. A logic "1" for 21 microseconds following the receipt of a valid data sync.

6. Take Data. A logic "1" during the time the NRZ data out is available from the MIU.

7. Valid Word. A logic "0" for 3.5 microseconds following the output of a valid word from the MIU via the NRZ data out line.

8. Valid Parity. A logic "1" for 500 plus or minus (TBD) microseconds following the output of a word with a valid one's parity from the MIU via the NRZ data out line.

9. NRZ Out. NRZ data presented to the PIU.

5.2.2.2 Performance

5.2.2.2.1 MIM General Characteristics. The MIM shall continuously monitor both the primary and secondary channels when in the receive mode. The MIM shall operate in the receive mode at all times except when requested to transmit data as indicated by the presence of a logic "1" transmit-receive bit of a valid command word having the applicable MIM address. After the MIM has responded, the MIM shall switch to the received mode within 1 microsecond. The switching shall be accomplished without generating transients on the multiplex channel.

5.2.2.2.1.1 Receive Mode. The major functions of the MIM in the receive mode shall be as follows:

1. Accept the incoming signal in the specified format.

2. Decode the incoming signals using the Manchester code to derive clocking.

3. Determine the validity of the incoming words by means of its seif test logic.

4. Determine the command word address is identical to the address assigned to the MIM. If the address is not identical, ignore the incoming signals until an applicable command word is received.

5. Decode the command word that has an address identical to the MIM's and provide a data block starting address/mode to the LRU electronics.

6. Transfer the incoming controller data words to the LRU electronics.

7. Transfer an appropriate response word to the controller after the data words have been transferred tc the LRU electronics.

5.2.2.2.1.2 Transmit Mode. The major function of the MIM in the transmit mode shall be as follows:

1. Accept the incoming signals in the specified format.

2. Decode the incoming signal using the Manchester code to derive clocking.

3. Determine the validity of the incoming command word by means of self-test logic.

4. Determine if the command word address is identical to the address assigned to the MIM. If the address is not identical, ignore the incoming signals until an applicable command word is received.

5. Decode the command word that has an address identical to the MIM's and provide a data block starting address to the LRU electronics for the transfer of data from the LRU to the MIM.

6. Transmit an appropriate response word to the controller.

7. Receive the number of data words specified by the command word from the LRU electronics.

8. Transmit the incoming LRU data words to the controller.

5.2.2.2.1.3 Response to Command Word. The MIM response to an applicable command word shall be a function of the command word and the MIM/LRU self-test logic:

1. The command word's number of words code shall specify the number of words to be transmitted/received.

2. The command word's transmit/receive (T/R) bit shall specify whether the MIM shall collect data words from the LRU electronics or transmit incoming data words to the LRU electronics.

3. The command words data block/mode code shall specify the location of the first word to be transmitted/received by the LRU electronics or command the LRU into specific mode/operation.

4. The logic state of command word bit 5, the T/R bit, and the MIM self-test logic shall determine the MIM operation in order to form the required message.

5.2.2.2.2 Multiplex Interface Unit (MIU) Characteristics

5.2.2.2.2.1 General. The MIU shall consist of a transmitter, detector, transmitter switch, sync generator, Manchester encoder, command and data word identification logic, Manchester decoder, data validation logic, parity generator, and miscellaneous logic to control operation. The MIU shall have the capability of generating command syncs as well as data syncs.

5.2.2.2.2.2 Transmitter Switch. The MIU shall include a transmitter switch. The transmitter switch shall isolate the transmitter from the multiplex channel during the nontransmitting mode of operation.

5.2.2.2.2.3 Sync Field Generator. The MIU shall include a sync generator. The sync shall be transmitted prior to each data word.

5.2.2.2.2.4 Manchester Encoder. The MIU shall include a Manchester encoder to encode the data words.

5.2.2.2.2.5 Word and Message Identification. The MIU shall have the capability of identifying incoming words as command words or data words based on the contents of the sync field preceding the word.

5.2.2.2.2.6 Detector. The MIU shall include a receiver-detector(s). The detector shall detect 1.0 megabit per second Manchester biphase coded data that is greater than the threshold level. The detector shall detect both the positive and negative excursions of the Manchester biphase coded data.

5.2.2.2.2.7 Manchester Decoder. The MIU shall include a Manchester biphase decoder. The detected data shall be decoded from Manchester biphase to NRZ digital binary, "1"/"0" logic level signals.

5.2.2.2.2.8 Reset. The MIU control circuitry shall include a "reset" signal. The reset signal shall reset all the PIU logic circuitry, except the response/status registers, and shall also be wired to the MIM-LRU interface for use by the registers, and shall also be wired to the MIM-LRU interface for use by the LRU.

5.2.2.2.2.9 Data Validation. The MIU shall have the capability of recognizing improperly coded signals, a data dropout, or excessively noisy signals occurring during the reception of a word and of producing signals indicating that a nonvalid word has been received.

5.2.2.2.2.9 Validation Criteria. The incoming data shall be evaluated on a bit-by-bit and overall word basis. Each word shall meet the following requirements in order to be valid:

1. The word shall be preceded by valid sync.

2. The word shall have 21 bits.

3. The word shall have the correct parity.

4. Each Manchester bit shall have a "1"/"0" or "0"/"1" sequence.

Failure of the incoming multiplex word to meet the above criteria shall cause the MIU to transmit the appropriate status signal to the PIU response/status register.

5.2.2.2.2.10 Parity Generator. The MIU shall include a parity generator. The parity generator shall generate odd "ones" parity. The parity generator shall contain logic which will permit the inhibit of the generation of parity for data words. The LRU's which provide data words, which include parity, shall provide a steady-state, 3.5-volt signal which shall inhibit the MIU parity generation for the data words. This inhibit parity shall not inhibit the generation of parity by the MIM for the response word.

5.2.3 Interface to and Control of the Multiplex System by the Burroughs Multiprocessor

5.2.3.1 General

An overall diagram showing the connection of the multiplex buses to the Burroughs Multiprocessor is shown in Figure 5-12. Connection to the multiprocessor is via the switch interlock with a multiplex interface and control module providing the interface between the multiplex buses and the switch interlock. The multiplex interface and control module is essentially treated as a device to the multiprocessor. Two multiplex buses are used with one serving as a backup; it is required that the multiplex interface and control module be capable of working with either bus.

The multiprocessor system is required to function as the controller of the bus. Therefore all commands for data transmission originate from the multiprocessor system. The multiplex interface and control module in Figure 5-12 provides the interface between the multiplex buses and the switch interlock; it also provides some control for the data and command transmissions that occur on the multiplex buses, the amount of this control is dependent on the design of this module. Regardless of the design of this module, the multiprocessor system retains ultimate control of the multiplex system. Several approaches to the design of the multiplex interface and control module will be given below.

5.2.3.2 Multiplex Interface and Control Module

The degree of control of the multiplex bus between the Multiplex Interface and Control module (MIC) and the multiprocessor system is dependent on the complexity of the MIC. This primarily effects the degree of involvement of an interpreter in controlling the actual data transmission process on a multiplex bus.

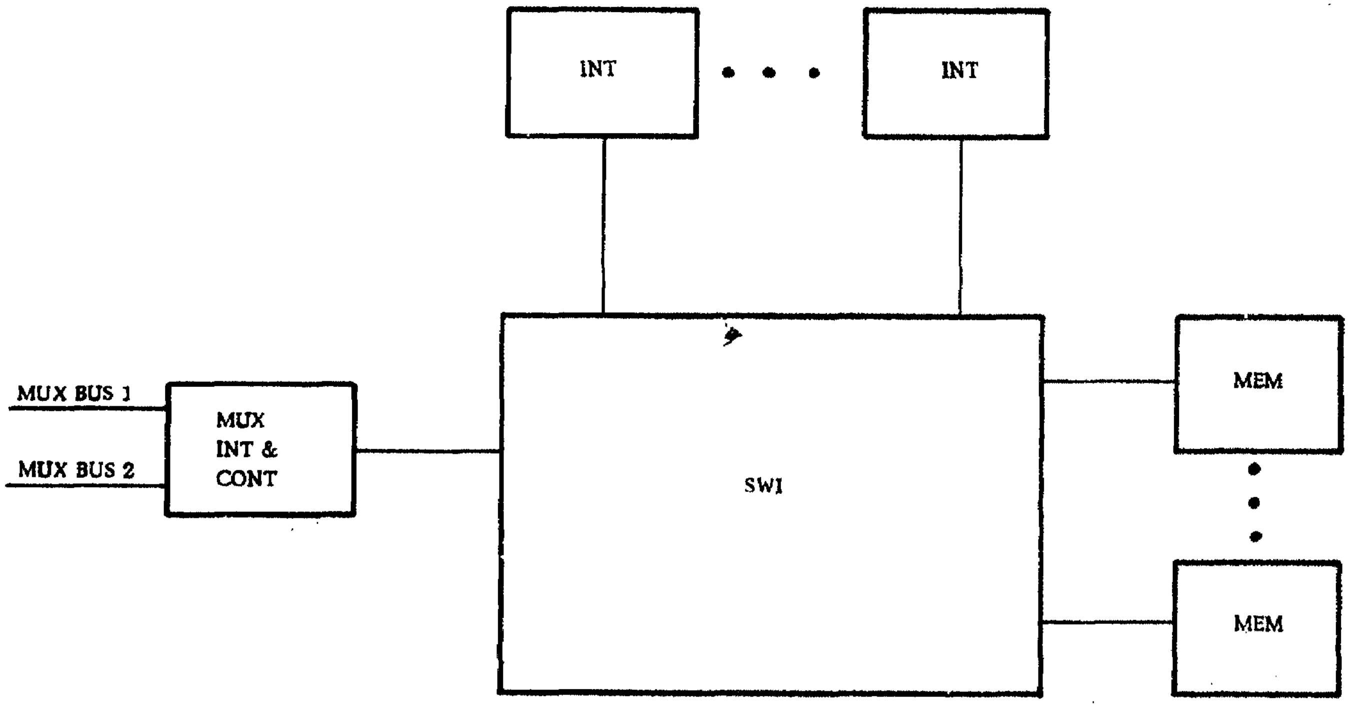


Figure 5-12. Connection of Burroughs Multiprocessor System to Multiplex System

5.2.3.2.1 Direct Interpreter Control System. A block diagram of the MIC with a minimum amount of control capability is shown in Figure 5-13. The left-hand portion of the figure contains functions (the MIU and serial-parallel registers) needed to interface with the multiplex bus. The MIU, multiplex interface unit, detects and generates sync codes, decodes biphase coded data, etc.; it is very similar to the MIU described in the previous section and the same unit may possibly be used here. The parallel/serial and serial/parallel registers provide the required serial interface for transmission on the multiplex bus. Two MIU's are used to provide capability for working with either multiplex bus as was explained in Section 5.2.2. The right-hand section of the figure is the interface that is required to work with the SWI. This consists of a set of input shift registers, a set of output shift registers, and clock interface logic. The information format at the SWI interface is shown in Figure 5-14. The interpreter sends commands and data (both 19 bits) to the MIC with the form shown in Figure 5-14 (c and d). In addition, two extra bits, for a total of 21, are sent to signify whether the information sent is to be output over the multiplex bus (command or data) or whether a request to input data to the interpreter is present. Information to be output on the multiplex bus is sent to the parallel/serial register and then to the MIU. The interpreter is responsible for timing the rate at which it sends information to the MIC. This rate should be high enough to supply a steady stream of output on the multiplex bus (24 usec per command/data word) and not too high such that the IN shift register is not cleared to accept information from the interpreter.

The MIC sends information to the interpreter with the format shown in Figure 5-14 (d). This consists of data and response words (20 bits). The MIC continuously loads this information into the OUT shift register as it is received and processed from the multiplex bus. The contents of the OUT shift register are sent to the interpreter when a "data in" request is received from the interpreter via the IN shift register. The interpreter is responsible for keeping up with the rate at which information is received over the multiplex bus.

The lower right-hand portion of this figure contains logic to receive the clock from the SWI (used to drive the shift registers) and logic to return the clock to the SWI. The counter counts four clocks and enables the readout of the shift registers. The control portion of the MIC will enable the return of clock signals to the SWI by the generation of an acknowledge signal. Note that the MIC could control the output rate of the interpreter into the IN shift register by not acknowledging if the parallel/serial register still contains information to be sent out on the multiplex bus.

5.2.3.2.2 Single Command Buffered System. The previous system required close operation of the interpreter with the entire transmission process on the multiplex bus. The interpreter would probably not be capable of doing any other functions while it is controlling an input or output transmission on the bus. In other words, while transmissions are in progress on the multiplex bus, an interpreter would be dedicated to an I/O processing function.

A slightly more complicated MIC is shown in Figure 5-15. This MIC has a small buffer memory added to it for the purpose of providing sufficient memory to process one command (up to 31 words) without continuous information transfer with the interpreter. In this system the interpreter sends a command to the IN shift register as in the prior case. However, the command is now examined to determine if a read or write to a subsystem is requested and the number of words in the

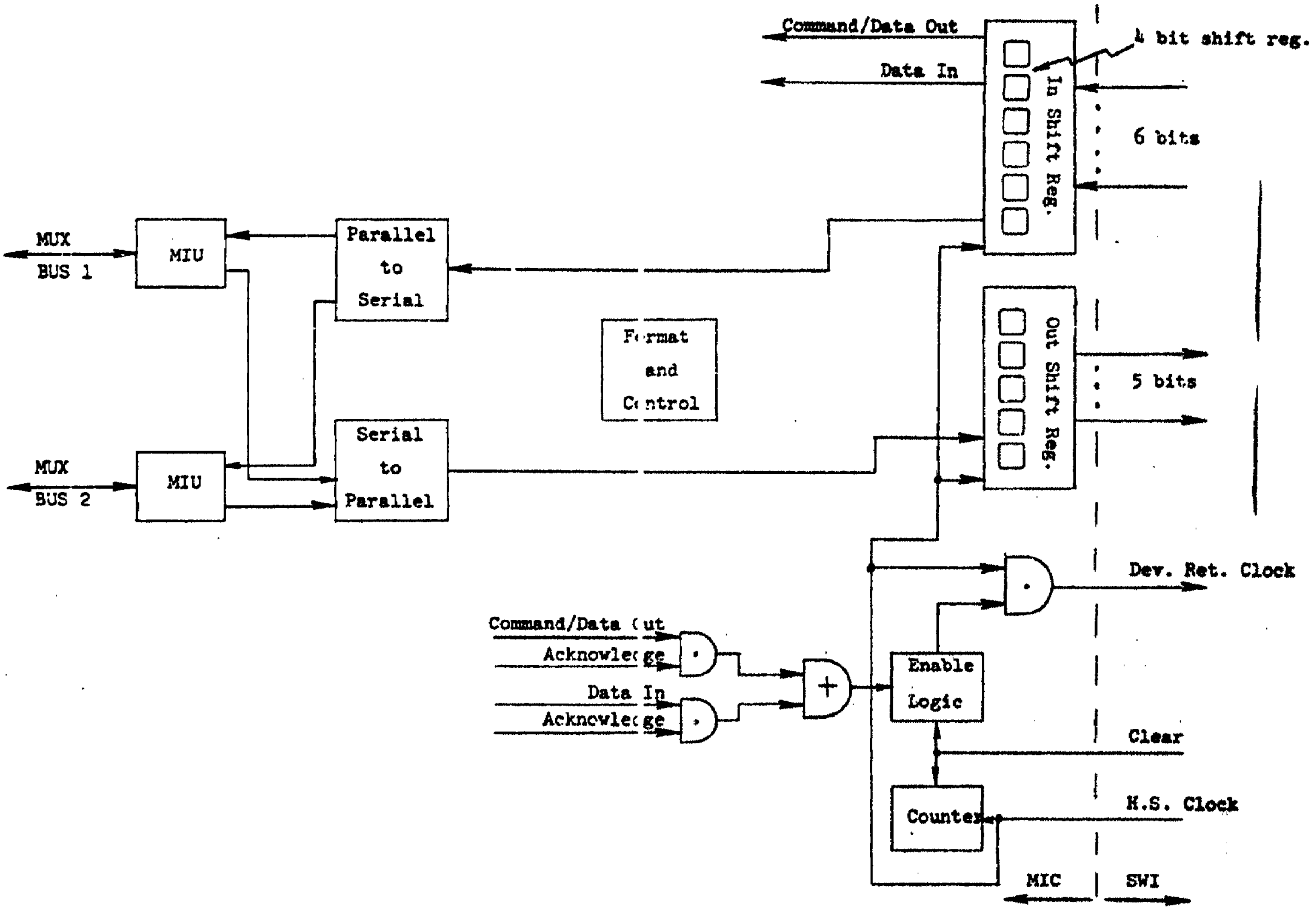
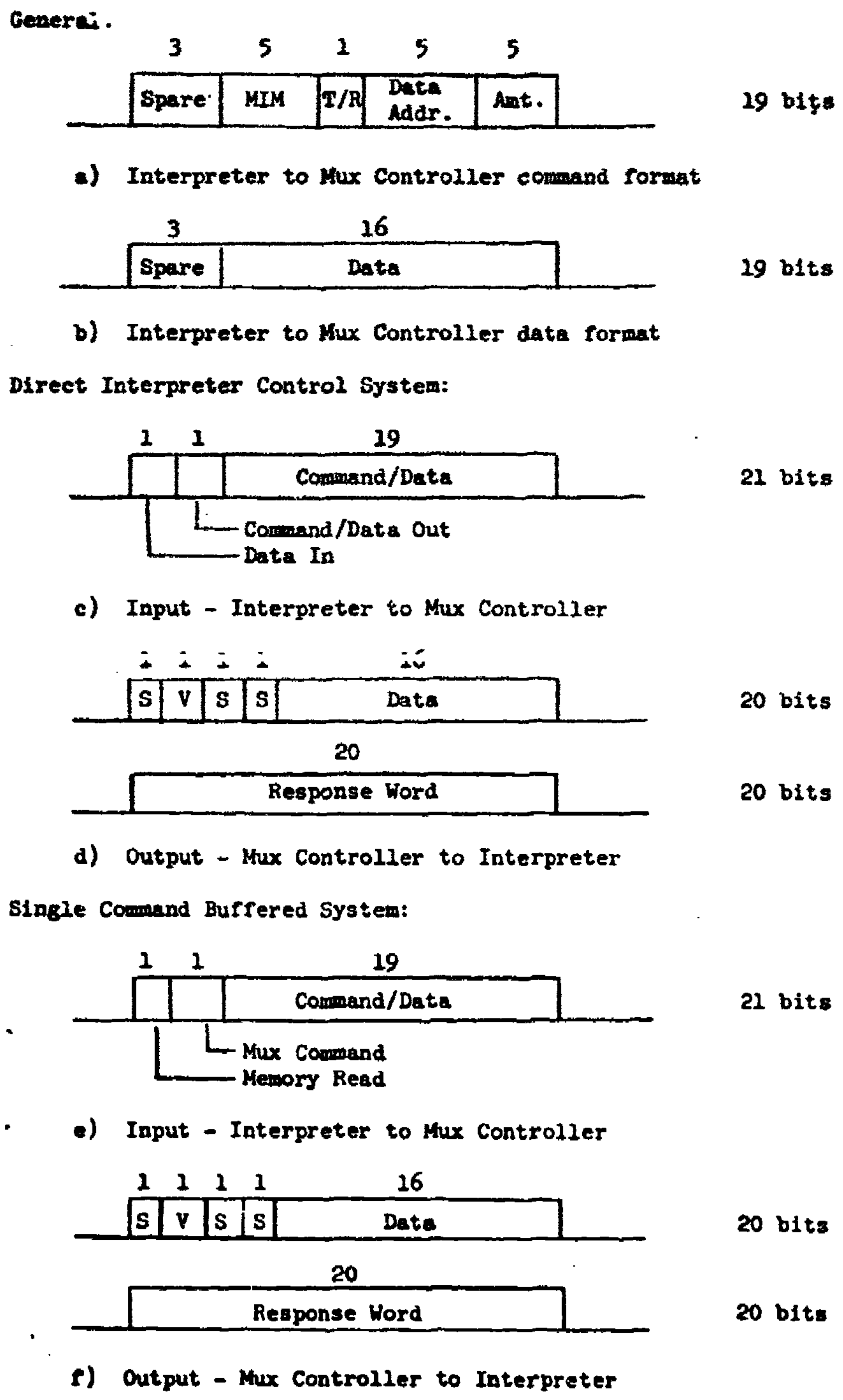


Figure 5-13. Direct Interpreter Control System



a) Interpreter to Mux Controller command format

b) Interpreter to Mux Controller data format

c) Input - Interpreter to Mux Controller

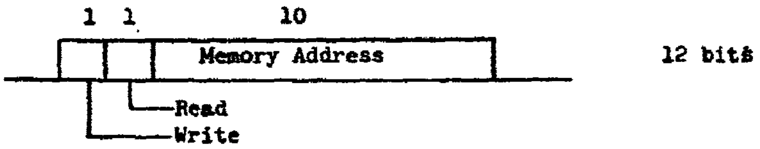
d) Output - Mux Controller to Interpreter

e) Input - Interpreter to Mux Controller

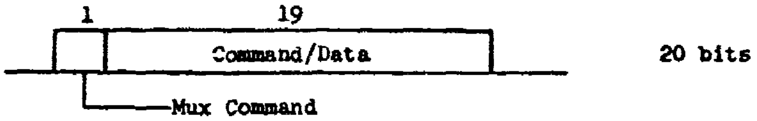
f) Output - Mux Controller to Interpreter

Figure 5-14. Data and Command Format for MUX Controller – Interpreter Interface

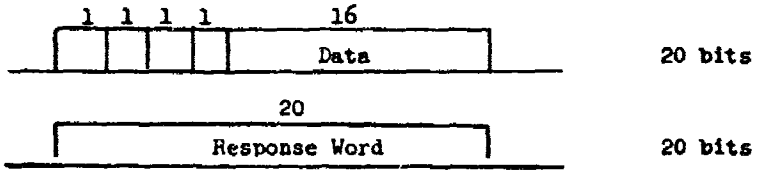
Multiple Command Buffered System:



g) Input - Interpreter to Mux Controller (Address Shift Register)



h) Input - Interpreter to Mux Controller (In Shift Register)



i) Output - Mux Controller to Interpreter

Figure 5-14. (Cont)

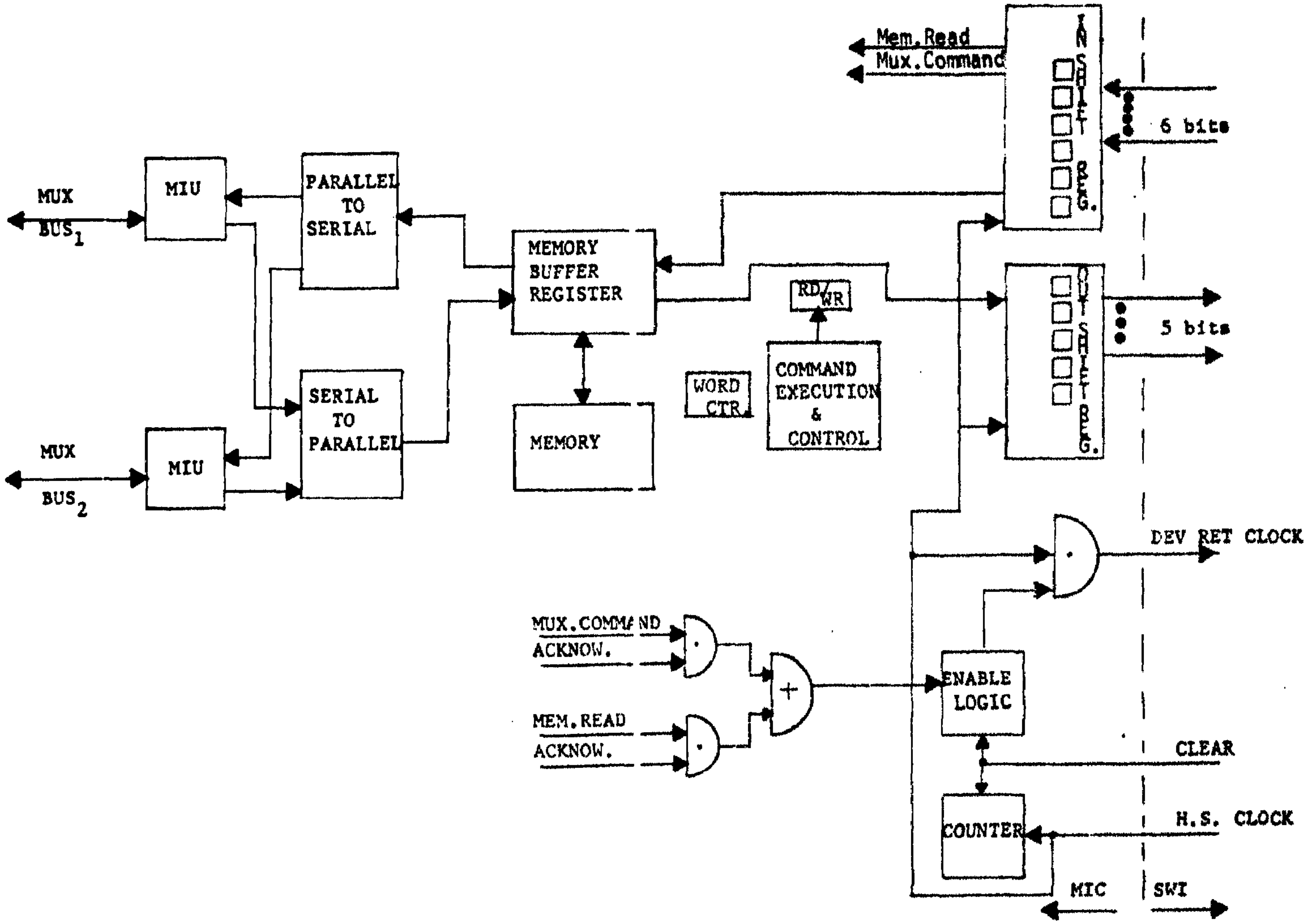


Figure 5-15. Single Command Buffered System

transmission. The command is output on the multiplex bus and the memory used to buffer either input or output data between the multiplex bus and the interpreter. The interpreter is now not tied to the information rate on the multiplex bus; it can load up to 31 data words in the memory as fast as it can for output or it can come back when it desires to read in up to 31 data words from the memory for input. The input format from the interpreter is shown in Figure 5-14 (e). Two control signals are used; one to indicate the presence of a command and the other to indicate a request for a memory read (note that one word at a time will be read out of the memory with the MIC word counter stepping through the set of buffered data in the memory).

5.2.3.2.3 Multiple Command Buffered System. A further increase in complexity of the MIC is shown in Figure 5-16. The previous system allowed the interpreter to issue one command to the MIC and then not be tied to the data rate on the multiplex bus. However, if a string of commands were to be processed the interpreter may or may not be able to do other meaningful work in between the commands. The MIC shown in Figure 5-16 allows the interpreter to do exactly what was done with the MIC in Figure 5-15 except that this MIC can process a string of commands so that the interpreter is further isolated from the transmission rate and message sequences on the multiplex bus.

In addition to data, commands are now also stored in the memorv. Additional logic is needed such that the MIC can sequence through a string of commands, recognize that all commands have been executed, and know where the data is to be stored in the memory. The interface with the interpreter is via an address shift register and an IN shift register as shown in Figure 5-14 (g and h). The address shift register receives an address (10 bits - should be adequate for the largest buffer memories used) and two control signals signifiying whether a read or write into that address is requested. The IN shift register receives 20 bits which consists of the command or data to be output and one bit signifying whether a command or data is present. An address of where to store commands in the memory is not sent over to the MIC with the commands since the MIC contains a command storage counter that points to an address of where to store the next received command. This counter simply recycles on itself. The command program counter points to the address of the command currently being executed by the MIC. When a command is received from the interpreter the command storage counter is used to store the command and it is incremented by one. The command program counter will continuously fetch commands as long as its contents do not equal the command storage counter. The address that is sent over with a command is the address where the multiplex bus response word should be stored in the memory. This address will have to be saved with the command.

The interpreter accesses information in the memory by sending the MIC an address with a memory read request. The MIC then interleaves this request with any command execution in progress and sends the interpreter the requested information via the OUT shift register.

The MIC contains control circuitry, data address counter, word counter, etc., to execute commands. One problem, that arises with dedicating more control to the MIC, is the identification by the interpreter of when a command has been executed. One means of providing such identification is via the response word.

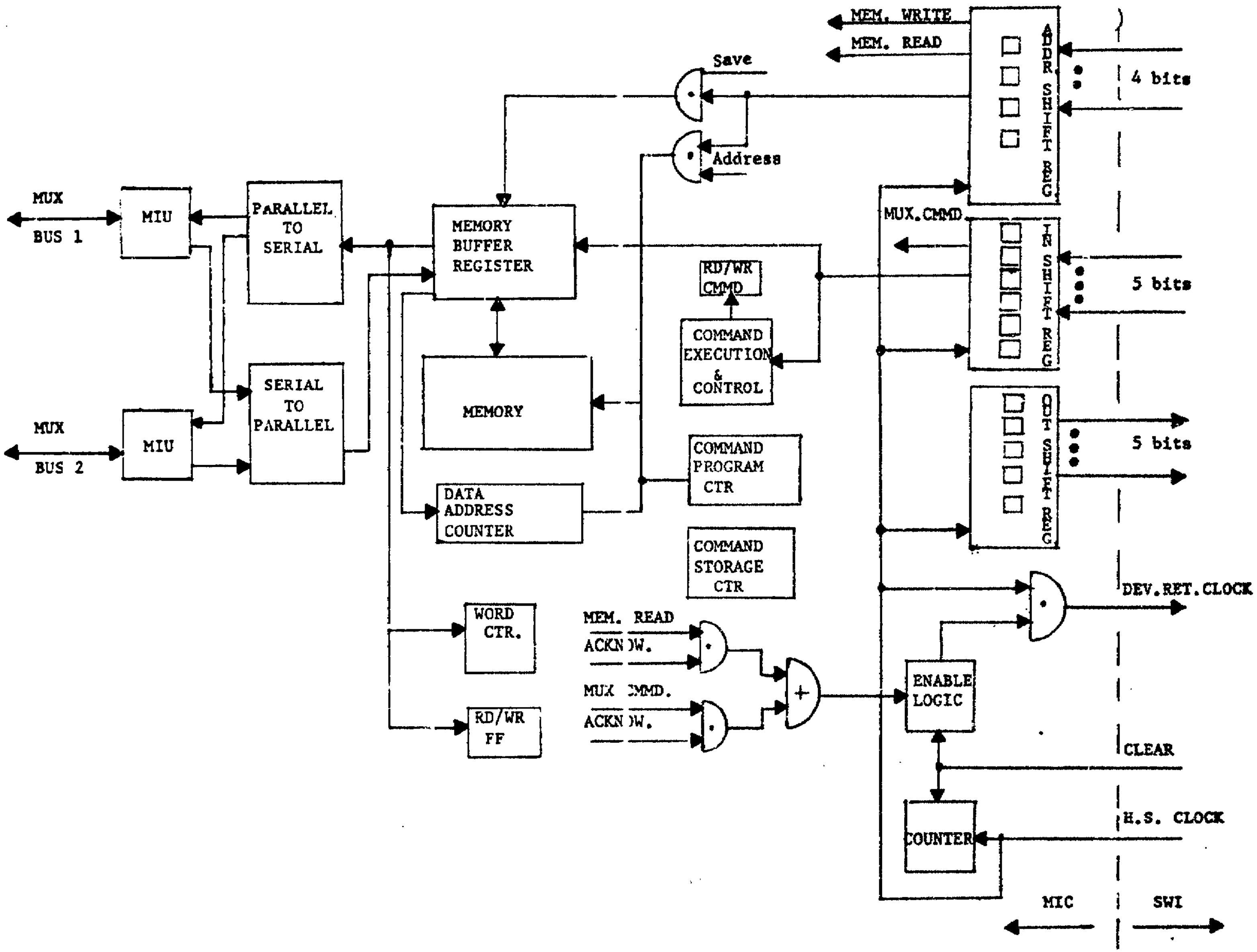


Figure 5-16. Multiple Command Buffered System

If the location that the response word is to be stored in is set to zero when a command is loaded in the memory from the interpreter, then this location can be checked by the interpreter to determine if a command has been executed. The setting of this memory location to zero could be performed by the interpreter or by the MIC.

This design allows the interpreter to set up a sequence of commands and output data in the MIC and then return to other processing tasks. The interpreter can then return later to fetch received information from the MIC.

5.2.3.2.4 Dedicated Interpreter System. Another possible approach to the MIC is to make it a true I/O processor by using an interpreter with its own memory; such an approach is shown in Figure 5-17. A Port Select Unit (PSU) is used to connect the interpreter to the memory since this interpreter essentially operates as a single interpreter.

The complete I/O program now resides in the MIC. The only interaction with the multiprocessor portion of the system is the transfer of data and possibly some form of master sync or control.

5.2.3.3 Interpreter I/O Operation

The first three approaches to MIC design require the interpreter to execute an I/O program that sends the appropriate data and commands to the MIC and fetches the appropriate data from the MIC. It is not the intent at this point to design this I/O program, however, it is observed that this area is a natural for macro "S" instructions. A single "S" instruction could be designed to fetch a command and data from main memory, send it to the MIC, check the response word, and do any retransmission in case of errors. A first cut at some of the gross steps involved in such a macro "S" instruction is given below (for data output to the MIC):

1. Fetch "S" instructions

2. Device locked? Lock if not

3. Execute device write to send command word to MIC

4. Fetch data from memory (two words/cycle)

5. Execute device write to send data to MIC

6. Execute device read to get response word from MIC

7. Check response word

8. Any errors? If so, retry

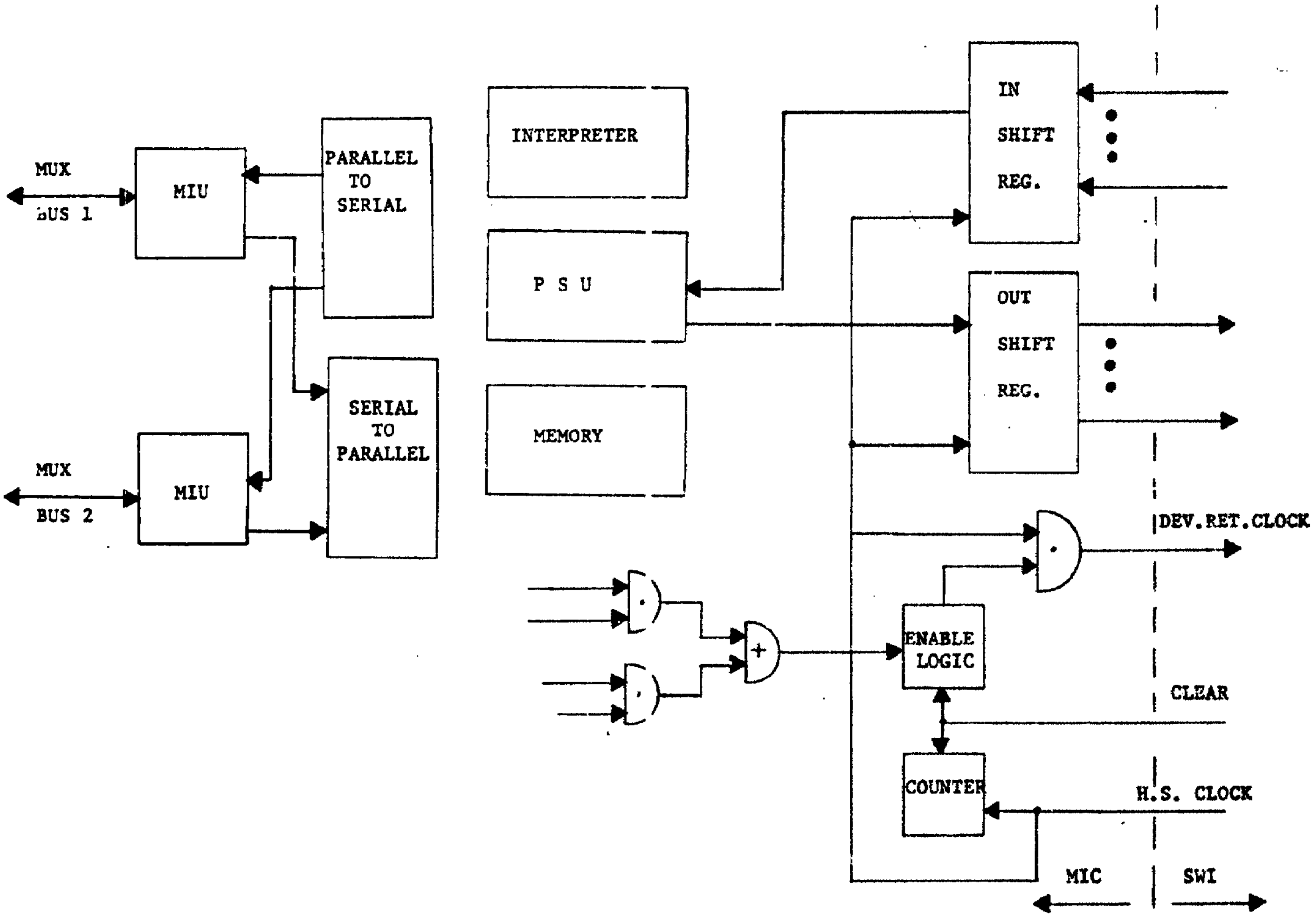


Figure 5-17. Dedicated Interpreter System

5.2.3.4 Conclusions and Recommendations

At this point an approach to the MIC cannot be firmly recommended. However, it is felt that the first two approaches, the direct interpreter control system, or the single command buffered system would be preferred. The reason is that the interpreter was designed to be flexible and easily configured to do arithmetic or I/O processing in a dynamic nature. The interpreter can very well do the I/O processing. In fact, an interpreter could be assigned to I/O processing only for the actual time needed; it may be assigned to other processing tasks when I/O processing is not needed. It is also desirable not to add specialized or complex hardware to the interpreter system since this hinders modularity and may result in problems from a failure tolerance standpoint. It will be assumed at this point that the first approach, the direct interpreter control system, which is the simplest MIC module, will be used in the central processor for purposes of this study.

5.2.4 System Configuration

5.2.4.1 System Description

The I/O requirements and the computer system configuration based on the processing allocation as shown in Figure 5-7 were reviewed to define the method of input/output for the entire system. A system configuration including the I/O implementation is shown in Figure 5-18. Four modules (or devices), each interfacing with SWI's, are shown for performing the I/O:

1. MIC - Multiplex Interface Controller

   Allows a multiprocessor to control the communications on a multiplex bus as described in Section 5.2.3.

2. PC - Parallel Channel

   Allows one multiprocessor to communicate with another multiprocessor. This is a dedicated channel and operates on a request acknowledge basis.

3. MT - Multiplex Terminal

   The basic interface for all devices or subsystems (this includes a multiprocessor that is part of a subsystem, e.g. the SRAM processor) connected to a multiplex bus. It recognizes a unique ID address and responds to a command sent out from the MIC to send or receive data over the multiplex bus. This module is functionally very similar to the MIM described in Section 5.2.2.

4. DI - Device Interface

   Interfaces subsystems or devices directly to a multiprocessor. The operation of this module is not controlled by the multiplex bus. It will typically be a specialized module matching the requirements of particular subsystems.

Two multiplex buses are used in this system. Both are driven by the central processor, with one bus dedicated to one multiprocessor and the other bus to the second multiprocessor. Each bus is actually dual redundant as explained in Section 5.2.2. Each multiprocessor is connected to its bus by a MIC module. Local processors, such as the IMU processor, and subsystems/devices are connected to each bus. All information transfer is to/from the central processor and under control of the central processor. The local processors and subsystems/devices interface to the multiplex bus via a MT module.

The central processor and pen aids processor are actually multicomputers, in that more than one multiprocessor is contained therein. Communication between these multiprocessors in a multicomputer configuration is via a PC module. An alternate approach would be to communicate via the multiplex buses; however, it is felt this method would require a similar amount of hardware and result in a slower communication link. In the central processor this alternate method would require the addition of MT modules to each multiprocessor. Use of the alternate method for the pen aids processor, would require the central processor to set up communication links between two multiprocessors in the pen aids processor.

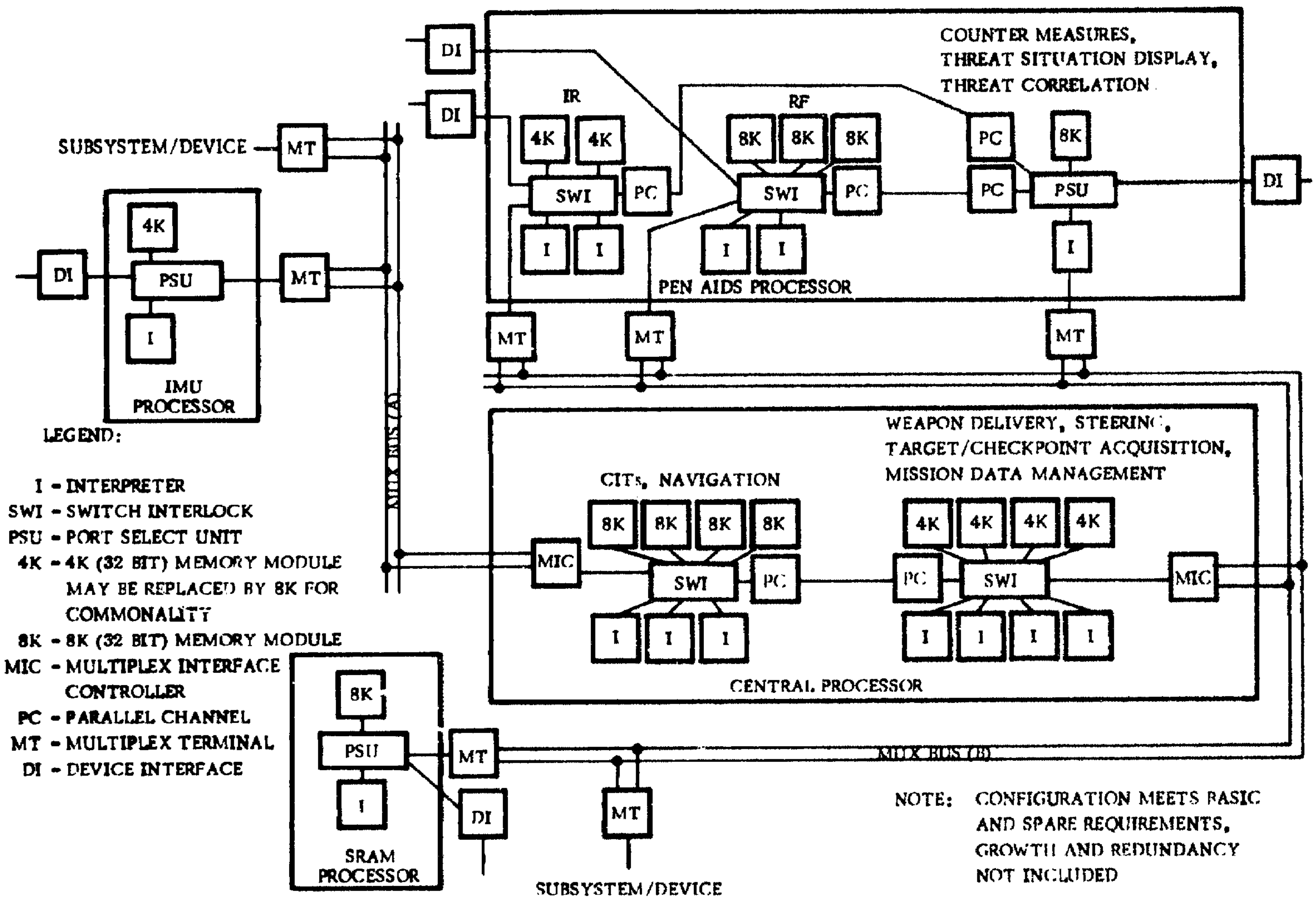


Figure 5-18. ASB Avionics Computational System Interface

The exact method of communicating with subsystems or devices that interface directly with the local processors is not completely specified at this time. The use of DI modules is shown in Figure 5-18. This actually could be a set of direct connections to subsystems/devices or a multiplex bus could be used as is the case for the central processor.

5.2.4.2 I/O Requirements and Processing Implications

A quantitative tabulation of the I/O requirements is given in Table 5-16. The rate, in words/sec, is given for each of the information links shown in Figure 5-18. These are basic data rate requirements and include the 50 percent spare requirement. These requirements are for information or data transfer and do not include any overhead for command words required to operate the various information links.

The method of operating the multiplex bus was described in Section 5.2.2. Two words, a command word and a response word, are required for each message. Each message can have up to 31 data words. The data itself is 16 bits, however, data and command words are transmitted as 24 bit words by the time sync, parity, etc., are added to each word. The bit rate on the bus is 1 MHz; therefore, a maximum of 41,500 words/sec can be sent on the bus (including data and commands) if any delays and dead time on the bus is ignored.

In practice, all messages will not be fully loaded with 31 data words. Examination of the I/O requirements reveals that a likely average for the number of words per message is 10. Adding two overhead words per message results in 5/6 efficiency on the bus. This results in an effective data or information rate on the bus of 34,600 data words/sec. Ad mentioned in Section 5.3 the simplest form of a MIC module will be assumed in this study, this basically dedicates an interpreter to performing I/O functions while I/O transmissions are in progress on the multiplex bus. Thus a measure of the percent of an interpreter's capability required to drive the MIC module can be determined by using the effective bus rate of 34,600 words/second and applying that to the data rate required with the MIC as shown in Table 5-16. It should be noted that this same capability requirement of an interpreter will apply to interpreters working with the MT module since the same effective data rate applies to that module.

The effect of the PC and DI data rates also needs to be considered on the required interpreter capability. Detailed designs of these modules have not been performed. However, it is expected a data rate of approximately 750,000 words/second shall be within the state of the art over the PC channel and data rates on the order of 100,000 words/seconds can be expected over the DI module.

Utilizing the I/O module data rates above and applying this to the requirements in Table 5-16, it is then possible to determine the percent of time an interpreter can be expected to be devoted to performing I/O processing functions. The results of this analysis are expressed in terms of the effective interpreter speed requirements in operations per second for performing I/O functions as shown in Table 5-16.

Having the I/O requirements on each processing configuration of Figure 5-7, it is now necessary to reexamine the configuration of Figure 5-7 to determine if it can handle the I/O processing functions. The largest I/O requirement is in the central processor, in particular multiprocessor 2. Examining the data in Section 5.1.3, shows that MP2 in the central processor contains sufficient capability to also handle the I/O

Table 5-16. I/O Requirements

Computer	I/O Data Rate Requirements (Words/Sec)*	Effective Interpreter Speed Requirements (Ops/Sec)
a. Ce:tral Processor		
1. Multiprocessor 1		
MIC	7,494	
PC	4,240	
Interpreter I/O Requirements		27,000
2. Multiprocessor 2		
MIC	19,549	
PC	4,240	
Interpreter I/O Requirements		58,520
b. Pen Aids Processor		
1. Multiprocessor 1		
MT	3,534	
PC	1,536	
DI	2,448	
Interpreter I/O Requirements		12,720
2. Multiprocessor 2		
MT	4,105	
PC	4,128	
DI	16,848	
Interpreter I/O Requirements		16,920
3. Multiprocessor 3		
MT	3,639	
PC	5,664	
DI	5,184	
Interpreter I/O Requirements		14,040
c. IMU Processor		
MT	1,860	
DI	3,111	
Interpreter I/O Requirements		6,240
d. SRAM Processor		
MT	210	
DI	8,520	
Interpreter I/O Requirements		1,800

* NOTE: 50 percent spare factor is included

processing functions. The same holds true for MP1 in the central processor as well as all the other multiprocessors in the ASB avionics configuration as shown in Figure 5-7. Therefore no additional interpreters are required to Figure 5-7 to accommodate the I/O processing requirements as defined in Table 5-16.

5.2.4.3 Interrupts

An additional factor that should be considered when investigating the I/O requirements is the need for interrupts. There are two basic types of interrupts to be considered, internal and external. At this point in time it is difficult to establish a quantitative figure for the number of these interrupts, however, both types will be required in this system. Internal interrupts typically consist of power up, parity error, storage protect, real time clock, etc.

The only external interrupt firmly identified at this time is the designate interrupt. The F-111 avionics had in addition to this interrupt an INS reset interrupt and two display freeze interrupts.

Therefore, it is felt that at least four external and four internal interrupts should be provided, a more conservative approach, and the recommended approach, would be to provide eight of each type of interrupt.

5.2.4.4 Other I/O Functions

In addition to the I/O functions described above, there will in all likelihood be two additional I/O functions when the final system is configured. One of these is a mass memory channel and the other is the provision of discrete input/output signals.

The ASB avionics system will have a mass memory, exactly how communications will be handled with this device is unknown at this time. There will probably be a device channel that allows communication with the mass memory.

In most avoinics systems there generally is some requirement that calls for several discrete I/O signals. An example of such a signal may be the failure go/no-go status of the computer. There should be some provision for handling several discrete I/O signals.

5.3 FAILURE DETECTION AND RECONFIGURATION FOR THE BURROUGHS MULTIPROCESSOR

5.3.1 Introduction

The approach to failure detection, isolation and reconfiguration is outlined below. Detailed system requirements are not available for those factors and the selected approach is based on certain assumptions. One assumption is that the computer system is required to survive at least one failure with a higher failure tolerance desireable. It is also assumed that it is required to provide continuity of certain functions during any reconfiguration process. The general approach to failure detection and reconfiguration is shown in Figure 5-19. Two multiprocessors are shown in this figure (this corresponds to the central processor in Figure 5-18), each one is comprised of a switch interlock with some number of memory and interpreter modules.

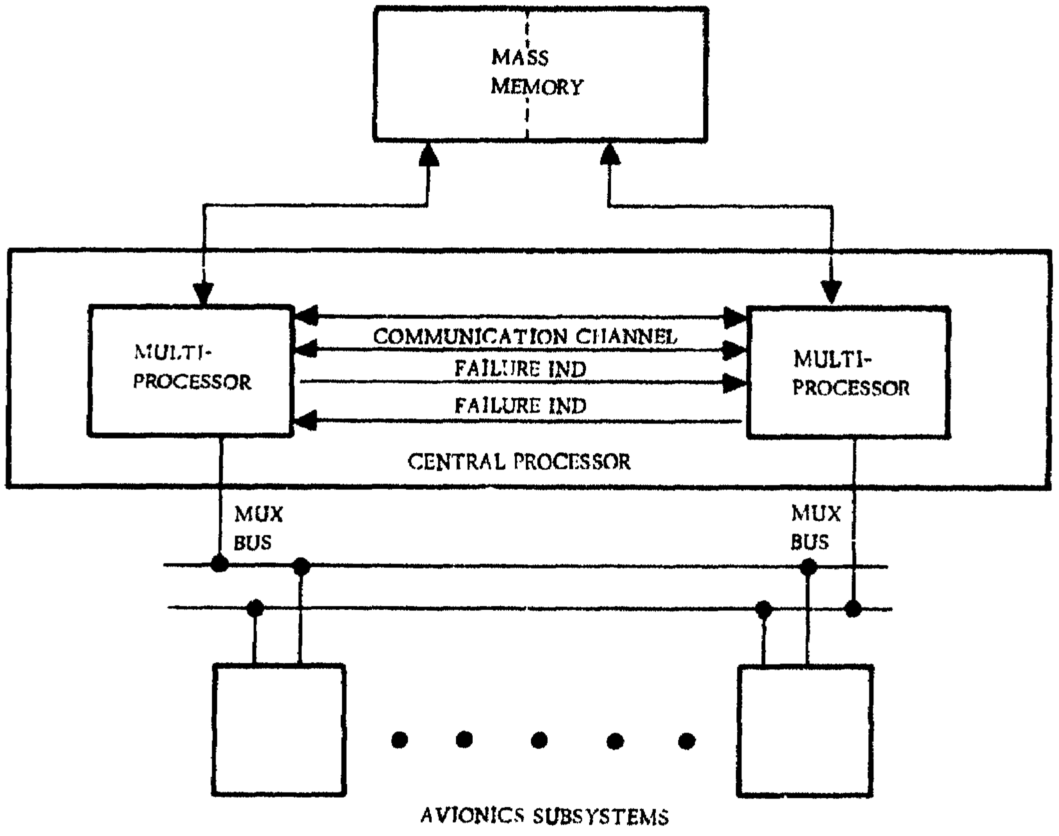


Figure 5-19. System Description for Failure Tolerance

This system is a multicomputer system with two computers where each one is a multiprocessor. Each multiprocessor contains its own failure detection capability and reports its failures to the other multiprocessor. Each multiprocessor contains backup capability for the other. When a failure indication is received, the non failed multiprocessor enters a backup mode wherein some of its normal functions may be suspended and some of the failed multiprocessor's functions are performed. The failed multiprocessor contains its own isolation and reconfiguration capability.

The multiprocessors each have their own dedicated mass memory. Depending on the computational functions, the mass memory may be used to store past values of data such that some computational functions may be restarted. Data integrity in the failed multiprocessor cannot be assumed after a failure and reconfiguration requires a complete restart or reloading data from the other multiprocessors or from mass memory.

This approach to failure detection and reconfiguration is very similar to that used for the F-111 Avionics System except that the system described here contains reconfiguration capability within each computer (a multiprocessor) of the multicomputer system (the F-111 System contained two single computers with no reconfiguration capability within a computer). The basic approach described in Figure 5-19 is applicable when two multiprocessors are used. Referring to Figure 5-18 in the previous section describing the interface to the ASB multiplex system, this is applicable to the 'central processor'. The approach to failure detection and reconfiguration for the IMU, SRAM, and Pen Aids processors would depend on the individual requirements of these sybsystems. The Pen Aids processor contains more than one multiprocessor and the same approach as described in Figure 5-19 could be applied here. The IMU and SRAM processor contain one multiprocessor. Reconfiguration within these multiprocessors is possible however computational continuity during reconfiguration cannot be assured unless more than one multiprocessor is used as described in Figure 5-19.

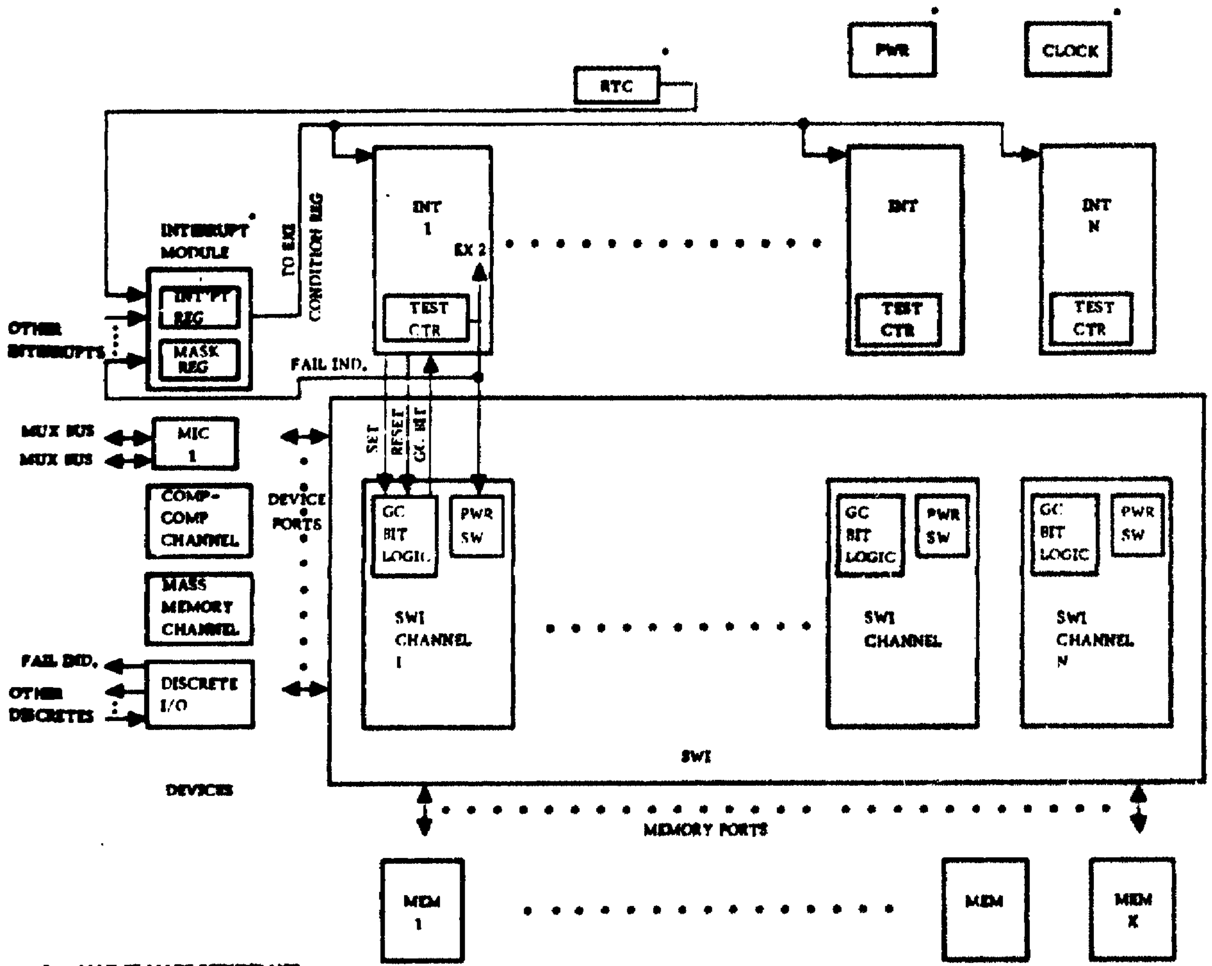
5.3.2 Modified Burroughs Multiprocessor

A block diagram indicating the modified version of a multiprocessor, configured from the Burroughs multiprocessor system, is shown in Figure 5-20. This modified multiprocessor incorporates a number of changes or additions to meet the ASB system requirements and to provide for failure tolerance.

5.3.2.1 Real Time Clock

A real time clock (RTC) has been added to the system. The real time clock is a counter that is driven by the system clock, counts down to zero which generates a signal that is input as an interrupt to all interpreters via the interrupt module, and is reset to a specified value (typically 1/64 or 1/32 of a second). The RTC is needed to provide a precise timing source for the execution of periodic programs in a real time mode. (The executive aspects are discussed in Section 5.4.)

In servicing this interrupt, each interpreter will attempt to access the system executive tables to determine what task to perform next. This will be accomplished by attempting to set the GC bit. Therefore, in case of exactly identical timing response to the RTC interrupt, the interpreter with the highest priority (as determined by the connections in the logic of the SWI Channels) will be the first one to access the executive. An alternate choice, to the approach shown in Figure 5-20 of providing a common RTC,



*: MAY BE MADE REDUNDANT

Figure 5-20. Modified Multiprocessor

is to provide a RTC per interpreter and use this RTC for an internal interrupt. The RTC in each interpreter would be driven by the common clock for the entire system. This would ensure that the RTC within each interpreter remains in sync. There appears little if any advantage to this approach and it ads hardware to an interpreter that may not be needed in other applications. Therefore, the approach of using a common RTC is recommended.

5.3.2.2 Modularity

The SWI has been redesigned to provide for better modularity and failure isolation as described previously in Section 4.3. The new approach basically partitions the SWI into channels where one channel is used per interpreter. The maximum number of interpreters and memories in a multiprocessor is limited to some amount, N and X, that is designed to initially. The SWI redesign kept X = 8 as was the case originally. The maximum number of interpreters can easily be designed to be greater than five which is the current figure. It should be noted that the figures for N and X include any spares that are to be reconfigured automatically (i.e. without any manual replacement or switching).

5.3.2.3 Interrupts

The interpreter as presently designed contains no true interrupts by the classical meaning of interrupts. Interrupts are provided for by having condition bits that can be tested via a microinstruction, the condition bits being set by an "interrupt". The microinstructions provide flexibility to perform this by allowing the successor microinstruction to be conditionally selected depending on the state of a condition bit. Therefore, the current microinstruction can state, for example, that the next microinstruction is arrived at via a jump if a certain condition is false or via a skip if a certain condition is true. Eight successor choices are provided for the false state and eight for the true state of a condition. A limiting factor in this approach, is that only one condition can be tested at a time.

The approach to mechanizing the interrupts is shown in Figure 5-20. An interrupt module is provided that basically 'or's' interrupts into one of the condition bits in each interpreter. The interrupt module contains an interrupt register which records which interrupt has occurred, the RTC is one of the interrupts, the failure indication from each interpreter provides an additional N interrupts, additional external interrupts as required by the system mechanization are input to this register. All of these interrupts are essentially or'ed to provide the signal to each interpreter that sets the EX1 condition bit in the interpreters.

The interrupt module is treated as a device (except for the signal to the EX1 condition bit which is hardwired directly) such that the interpreters communicate with it via the SWI. The interpreters can sample the interrupt register to determine what the interrupt was. This register will be reset when sampled. The first interpreter responding to the interrupt will sample the register and record the interrupt in memory. The remaining interpreters will sample a clear interrupt register and will then go to memory where the interrupt is stored and being serviced via the executive routine.

The EX1 condition bit will be tested in the last microinstruction of an S instruction. (If an S instruction were tested before completion, some means of saving the interpreter registers would be needed, no such paths exist in the interpreter at present.) The testing of this single condition bit thereby provides a test of all the interrupts input to the interrupt module. Other types of interrupts such as memory parity or memory protect, if provided, can be serviced by the other two external condition bits and would be tested in appropriate microinstructions (.e.g. memory protect would be tested after a memory write microinstruction.

The interrupt module also contains a mask register that may be set to any bit pattern by the interpreters. This register effectively masks selected interrupt lines into the interrupt module, it prevents these masked inputs from entering the interrupt register and setting the EX1 bit, if desired. This mask register can be used to prevent failed subsystems from hanging up the multiprocessor by continuously generating interrupts.

5.3.2.4 Global Condition Logic

In the course of investigating the failure detection, isolation and reconfiguration characteristics, it was determined that the global condition (GC) logic, as presently implemented, presents a single point of failure around which reconfiguration cannot be accomplished. Examination of Figure 7 of Reference 4 indicates that the request, GC bit, and ripple logic are all in series. Therefore, the failure of any interpreter in this path will disable the functioning of the GC logic in all interpreters. This problem cannot be solved by turning power off to an interpreter. In fact, this design approach prevents turning power off to an interpreter if it is not needed in operation.

An alternate approach to the GC logic is shown in Figure 5-21. This approach is similar to the Burroughs approach except that the GC request is sent out on a separate signal for each interpreter instead of being rippled through interpreters. The advantage of this approach is that power can be turned off to any interpreter without affecting the operation of the GC logic in other interpreters.

It was decided in the failure tolerance analysis that it would not be necessary to turn power off to interpreters. Therefore, in order to prevent a failed interpreter from affecting the operation of the GC logic, the GC logic shown in Figure 5-21 was moved to the SWI as shown in Figure 5-20 (power will be turned off to the SWI channels in the event of failure). The set, reset, and GC bit signals are sent between the interpreter and the SWI channel of that interpreter. It should be noted with this approach that the maximum number of interpreters, N, must be planned for ahead of time in the or logic of the GC request signals as shown in Figure 5-21.

5.3.2.5 Failure Detection, Isolation, and Reconfiguration

Several minor modifications were made to provide for failure tolerance. The GC logic was changed and moved to the SWI as explained above. A test counter was added to each interpreter. This counter must be periodically reset by the interpreter or a failure indication signal will be generated. A power switch was added to each SWI channel in the event a failure indication is generated within that interpreter. The SWI was redesigned on a channel per interpreter basis so that failures do not affect more than one interpreter. In addition software which allows tests of the memory, SWI and interpreters was added. Spare memory, SWI channel, and interpreter modules would be provided as needed to meet system reliability requirements. The approaches to failure detection, isolation, and reconfiguration will be explained in further detail below.

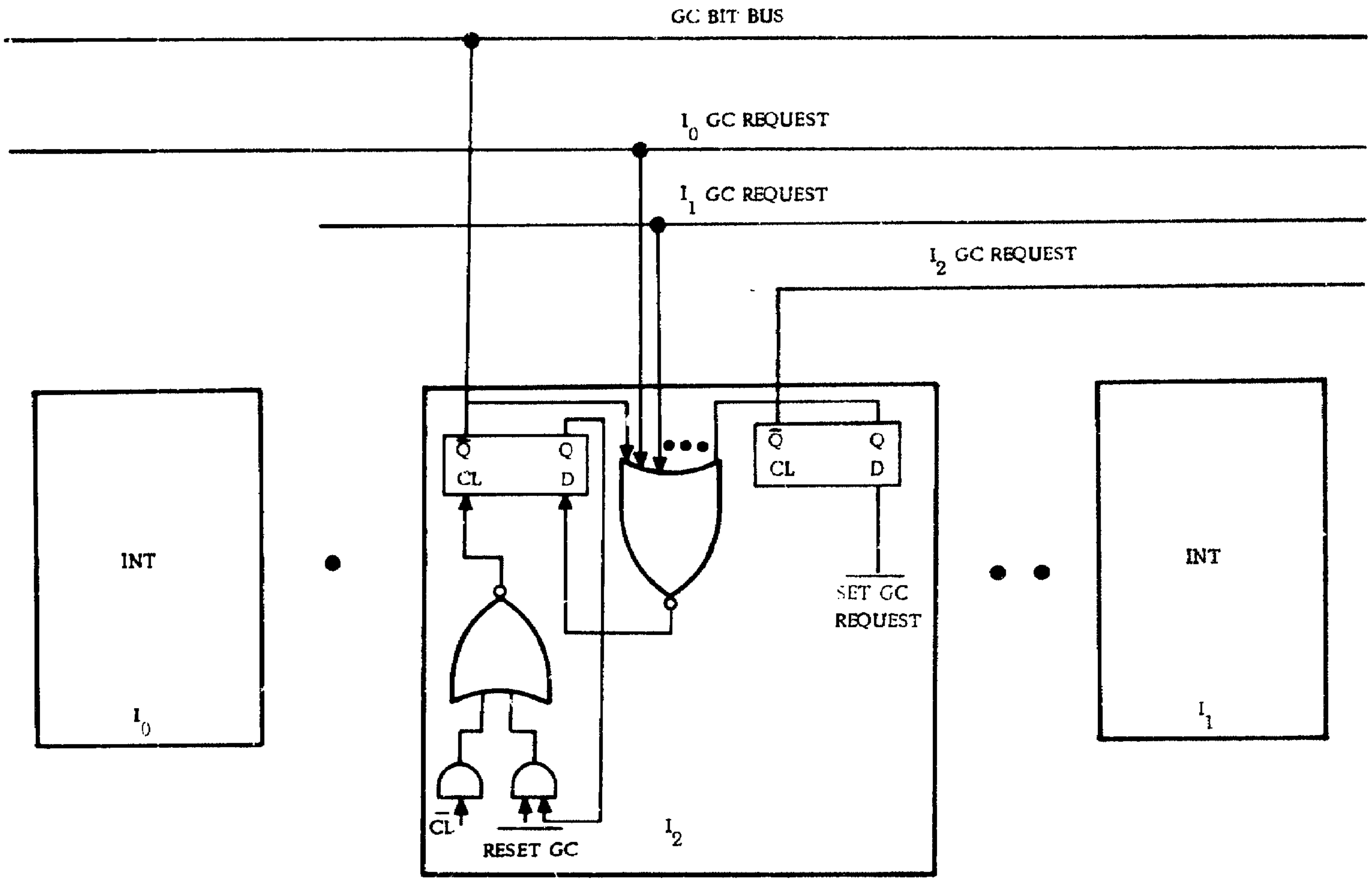


Figure 5-21. Modified GC Logic

5.3.3 Failure Tolerance

The general philosophy of the selected approach to failure tolerance was outlined in Section 5.3.1. The implementation of failure tolerance will be described in this section with a specific discussion of failure detection, isolation, and reconfiguration in subsequent sections. Failure tolerance requires that the multiprocessor, as shown in Figure 5-20, detects a failure within the multiprocessor, reports the failure to a backup device (another multiprocessor in the central computer case), and reconfigures into a correctly operating multiprocessor, informing the backup device of a successful reconfiguration.

The multiprocessor was investigated to determine if it could be relied upon to perform reconfiguration such that a backup device is not required. The basic problem in the multiprocessor is that data integrity cannot be assumed after a failure is detected. An interpreter, through a SWI, can access any memory module. Failures may occur such that the interpreter writes into the wrong memory location, thereby destroying data. Such failures can readily occur in many portions of the interpreter and SWI, e.g. in the address decoding of the SWI which selects the proper one out of eight memory modules, in the interface between the interpreter and the SWI, in the registers in the interpreter, in the data paths in the interpreter, in the adder of the interpreter, etc. These types of failures may also occur in the memory where incorrect information is received from one memory that results in a read into another memory module into an incorrect location.

Software and hardware schemes may be used to reduce the probability of such failures. e.g. coding on portions of the memory address logic (programs and constants may be preserved using various memory write protect schemes). However, in order to provide data integrity with a reasonable confidence level, after a failure, for a real time control application such as the ASB avionics, it is necessary to use massive redundancy with independent modules. In the multiprocessor, this would require the ability to operate the interpreters and memories as sets of independent computers. Some means of a lock mechanism would be needed to accomplish this such that a memory module could be dedicated to only a selected interpreter(s), preventing access (at least write) to all non selected interpreters. A system such as this could then be reconfigured after a failure by changing the control to the lock mechanism. It should be noted that as reliability requirements are increased to the point where any failure must be tolerated, then perfect failure detection, isolation, and reconfiguration must be provided, such stringent requirements require at least a level of redundancy of three (Ref 17), using techniques such as majority voting, to meet the reliability goals.

The approach taken here was to preserve the basic architecture of the Burroughs Multiprocessor as shown in Figure 5-20, allowing it to be used as a true multiprocessor. Data integrity is not assumed after a failure is detected in the multiprocessor. Redundancy in the form of another multiprocessor, as shown in Figure 5-19, is used during reconfiguration to achieve failure tolerance for those computational functions that require continuity of performance (this may be a degraded mode of performance). Failure tolerance for the ASB avionics central computer thereby takes the form of using two multiprocessors, each multiprocessor is reconfigurable after a failure, and each multiprocessor provides backup for the other during reconfiguration for critical functions requiring continuity of performance.

Each multiprocessor contains its own failure detection, isolation, and reconfiguration capability. Failure detection is primarily accomplished by using a software test routine that is scheduled for periodic execution by each interpreter. A test counter is provided for each interpreter, as shown in Figure 5-20, that must be periodically reset otherwise a failure indication signal will be sent out from the counter. The test counter is reset only if the software test routine is completed successfully. Failures detected by this approach can be caused by malfunctions in the interpreter, SWI, or memory modules and as stated above the failures could have destroyed data anywhere in memory (programs and constants may be preserved using a memory protect scheme on such information). Failures are not isolated between an interpreter and its SWI channel since loss of either precludes use of the other. A power switch, that is driven by the test counter in each interpreter, is provided in each SWI channel. This switch prevents a failed interpreter/SWI channel from affecting the proper operation of the multiprocessor after a failure is isolated and reconfiguration accomplished. Reconfiguration is accomplished by reloading a spare memory module, if required, with a copy of a failed memory module's program, reinitializing any required data, and informing the other multiprocessor of a successful reconfiguration.

5.3.4 Failure Detection

An overall flow chart depicting the failure detection process is shown in Figure 5-22. The failure detection program is scheduled to be executed periodically once every n seconds. It is entered by means of the RTC interrupt which sends the interpreters to the task scheduling tables. The failure detection task is scheduled for execution by each interpreter.

Part of the failure detection program resides in main memory and part in the interpreter's microprogram memory (MPM) (permanently in a RCM portion of the MPM). The portion of the failure detection routine that permanently resides in the MPM acts as an executive and controls the execution of the failure detection program.

As shown in Figure 5-22, Block 1, scheduling of the failure detection program results in control being transferred to a fixed location in the MPM of the interpreter. The first portion of the failure detection program checks the operation of the interpreter. The interpreter will fetch an interpreter test routine, which is part of the failure detection program, from main memory (Block 2). This routine will be checked to determine its integrity by forming a check sum of the routine. The interpreter will compare the results of the check sum with a permanently stored constant in the MPM. If the test does not agree, then another memory module (Block 4) is accessed for the interpreter test routine. After the test routine is validated, it is executed by the interpreter. This routine will check the logic, control and data paths in the interpreter. If the interpreter fails to execute this routine correctly, it will halt and the test counter will run out resulting in a failure indication.

Successful performance of the interpreter test routine will be followed by the interpreter testing its switch interlock (SWI) channel. The interpreter fetches a SWI test routine and determines its validity by means of a check sum (Blocks 8, 9, 10) in the same manner as for the interpreter test routine. The memory modules will each have a prestored constant in a known location which will be a different value and location in each memory module. The SWI test routine will read the constants and compare these with the expected response (Blocks 11, 12, 13). This test will check the information transmission paths and address decoding logic of the memory request section of the SWI. The device request section of the SWI is checked in a similar manner. The exact

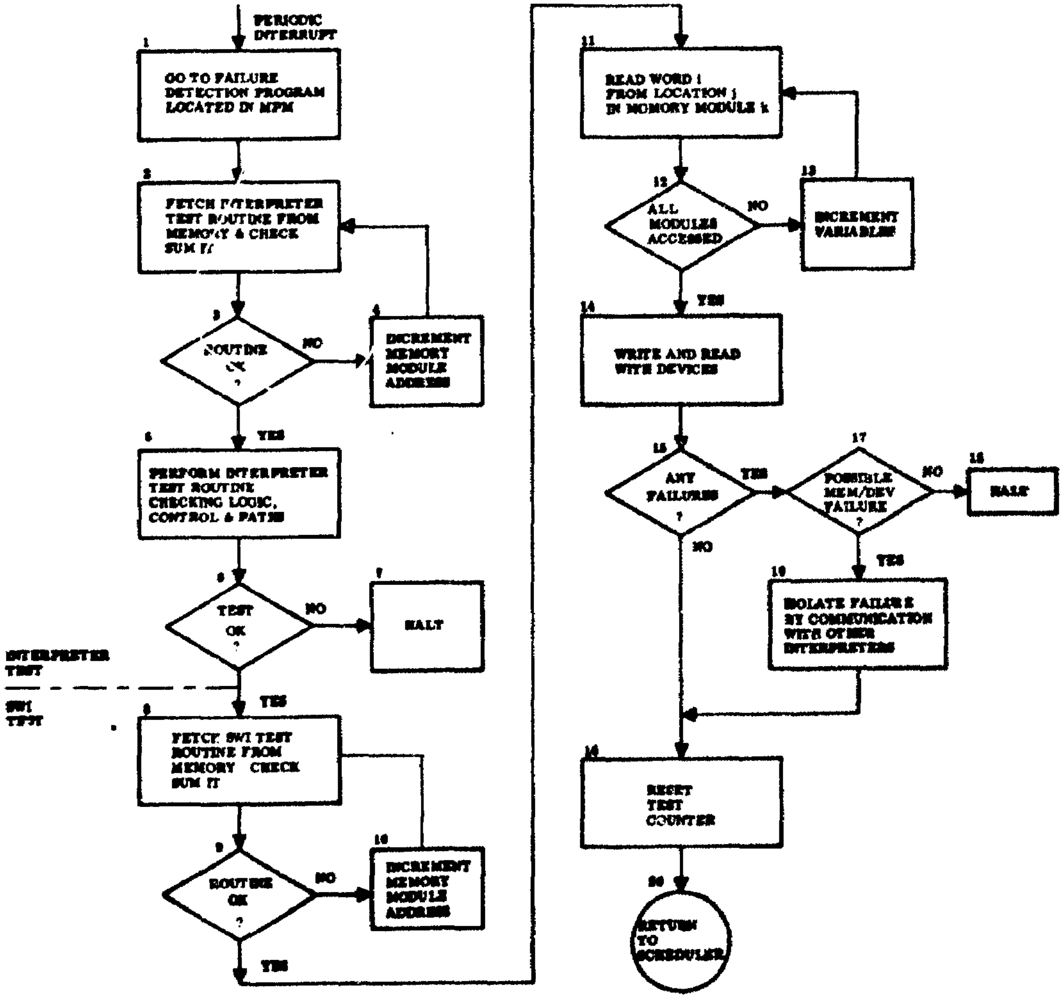

PERIODIC INTERRUPT
1
GO TO FAILURE DETECTION PROGRAM LOCATED IN MPM
2
FETCH INTERPRETER TEST ROUTINE FROM MEMORY & CHECK SUM IT
3
ROUTINE OK ?
NO
4
INCREMENT MEMORY MODULE ADDRESS
YES
5
PERFORM INTERPRETER TEST ROUTINE CHECKING LOGIC, CONTROL & PATHS
6
TEST OK ?
NO
7
HALT
INTERPRETER TEST
SWI TEST
YES
8
FETCH SWI TEST ROUTINE FROM MEMORY CHECK SUM IT
9
ROUTINE OK ?
NO
10
INCREMENT MEMORY MODULE ADDRESS
YES
11
READ WORD i FROM LOCATION j IN MEMORY MODULE k
12
ALL MODULES ACCESSED
NO
13
INCREMENT VARIABLES
YES
14
WRITE AND READ WITH DEVICES
15
ANY FAILURES ?
YES
17
POSSIBLE MEM/DEV FAILURE ?
NO
18
HALT
YES
19
ISOLATE FAILURE BY COMMUNICATION WITH OTHER INTERPRETERS
NO
16
RESET TEST COUNTER
20
RETURN TO SCHEDULER

Figure 5-22. Failure Detection Process

test used will depend on the characteristics of the devices connected to each device port, e.g. the interrupt module can be used to write and then read a value into the mask register, the MIC can be used to send a value to a subsystem and then retrieve it, etc. If the SWI is checked out with no apparent failures, then the test counter in the interpreter is reset.

If a failure is detected in the SWI test, it must then be isolated. The failure could be in a memory/device or in the SWI. Failures with more than one memory/device will be cause to suspect the SWI and failures with all memories/devices will result in the SWI being declared faulty in which case the interpreter would halt (Block 18) with the test counter eventually indicating a failure. Failures with only certain memories/devices will be isolated by communication with other interpreters to determine if they are also experiencing failures with those memories/devices.

Following successful completion of the interpreter and SWI channel tests the interpreter will return to the scheduler in main memory. The next test to be performed is on the memory modules. The memory module test program needs to be executed by only one interpreter. The first interpreter to complete its interpreter and SWI channel tests will execute the memory module test program (scheduled just like any other task) and the remaining interpreters will be assigned productive computational tasks. The memory module test program will test each memory module's ability to read/write selected bit patterns into selected locations.

The method used for failure detection does not guarantee that the failure indication from the test counter results in an isolated failure. The failure indication could be the result of another interpreter's failure or a memory module failure. This results from the fact that the failure indication program is scheduled via information contained in memory. This information could be incorrectly altered by a faulty interpreter or the memory itself could fail. This may result in the failure detection program not being scheduled in time to prevent the test counter from running out. An alternate approach, which may eliminate the above problem, and result in a failure indication that provides an isolated signal, is to use a self scheduling mechanism within the interpreters for the failure detection program. The test counter as shown in Figure 5-20 is an X bit counter. If an additional output from the counter is provided at t bits, then this output could be used to interrupt the interpreter and force the failure detection program to be performed in exactly the same manner as described in Figure 5-22. The length X-t would be selected to provide sufficient time to perform the failure detection program.

This method may result in executive scheduling problems since the interpreters would be scheduling tasks in addition to the tasks scheduled via the system executive tables. Timing and interference problems may result with this approach. Therefore, the first approach described is recommended at this time with the result that the failure indication signal from the test counter does not necessarily represent an isolated failure.

An additional point that should be noted here is the possibility of a memory resulting in a failure indication from an interpreter due to a no response failure. The interpreter interface with the memories is asynchronous with the interpreter requesting a memory operation and a memory response arriving at some variable time due to the multiprocessing operation of the system. The interpreter can halt and wait for the response, do other functions and intermittently test for a memory response, etc. A memory failure, such that no response is received, will eventually hang up the interpreter.

Burroughs used a time out counter in their bread board system to get around this problem. The timer was automatically reset every time a memory request was made and if no response was received the timer would overflow and set an external condition bit in the interpreter and also force the interpreter to step to the next MPM location.

It is not certain that this timer will be required in the ASB avionics application. A no response from the memory will eventually be detected by the test counter in the interpreter running out with a resultant failure indication signal. This is another case where the failure indication signal does not represent an isolated interpreter failure. One situation where a memory response timer is required, is during initial startup or failure isolation/reconfiguration in which the failure detection program as described in Figure 5-22 will be performed. Whenever main memory accesses are made (such as Block 2, 8, 11), some form of a timer must be used to prevent a memory failure from preventing the failure detection program to be run. This can be accomplished by using the CTR register in the interpreter for timing out the memory response. The interpreter can go into a small loop where this register is incremented and tested; if it overflows, the interpreter can branch out and not be hung up waiting for a failed memory.

This CTR register could possibly be used during normal operation to time out the memory response. However this is not certain at this time, since it may be required to implement the 'S' instruction and hence could not be relied upon to be available as a timer. It should also be noted that, if an isolated failure indication signal is required from the interpreter, such as was described previously by using a test counter with two taps on it, then some form of a memory response timer must be used.

The failure detection process described above required one hardware modification to the present system, the test counter. This counter will be driven by the interpreter clock and count up to a sufficient value (on the order of one second). The counter can be reset by using one of the 16 combinations provided by nanobits 51-54, Mem-Dev Op, since several spares presently exist. The counter could actually be placed in the MDC portion of the SWI, since these particular nanobits are decoded therein. In this case no change is actually made to the interpreter. However, if the counter is placed in the SWI, power to the counter must not be turned off by the power switch of the SWI since there would be no way to start up an interpreter from a cold start or after a failure indication for purposes of failure isolation.

5.3.5 Failure Isolation

An overall diagram depicting the failure isolation/reconfiguration process is shown in Figure 5-23. This diagram presents a first level overview of the process, numerous details are not shown here. The process is entered by a failure indication interrupt. The first step the interpreter takes is to reset the test counter and go to Block 2 which is basically the failure detection process which was described in the previous section and shown in Figure 5-22. The failure detection interrupt is broadcast to all interpreters and all interpreters will be following the process shown in Figure 5-23. It should be noted that the interpreter that sent out the interrupt cannot retrieve the actual interrupt from the interrupt register through the SWI channel, since its test counter will have turned power off, through the power switch, to its SWI channel. Therefore, as shown in Figure 5-20, the signal from the test counter is also wired to

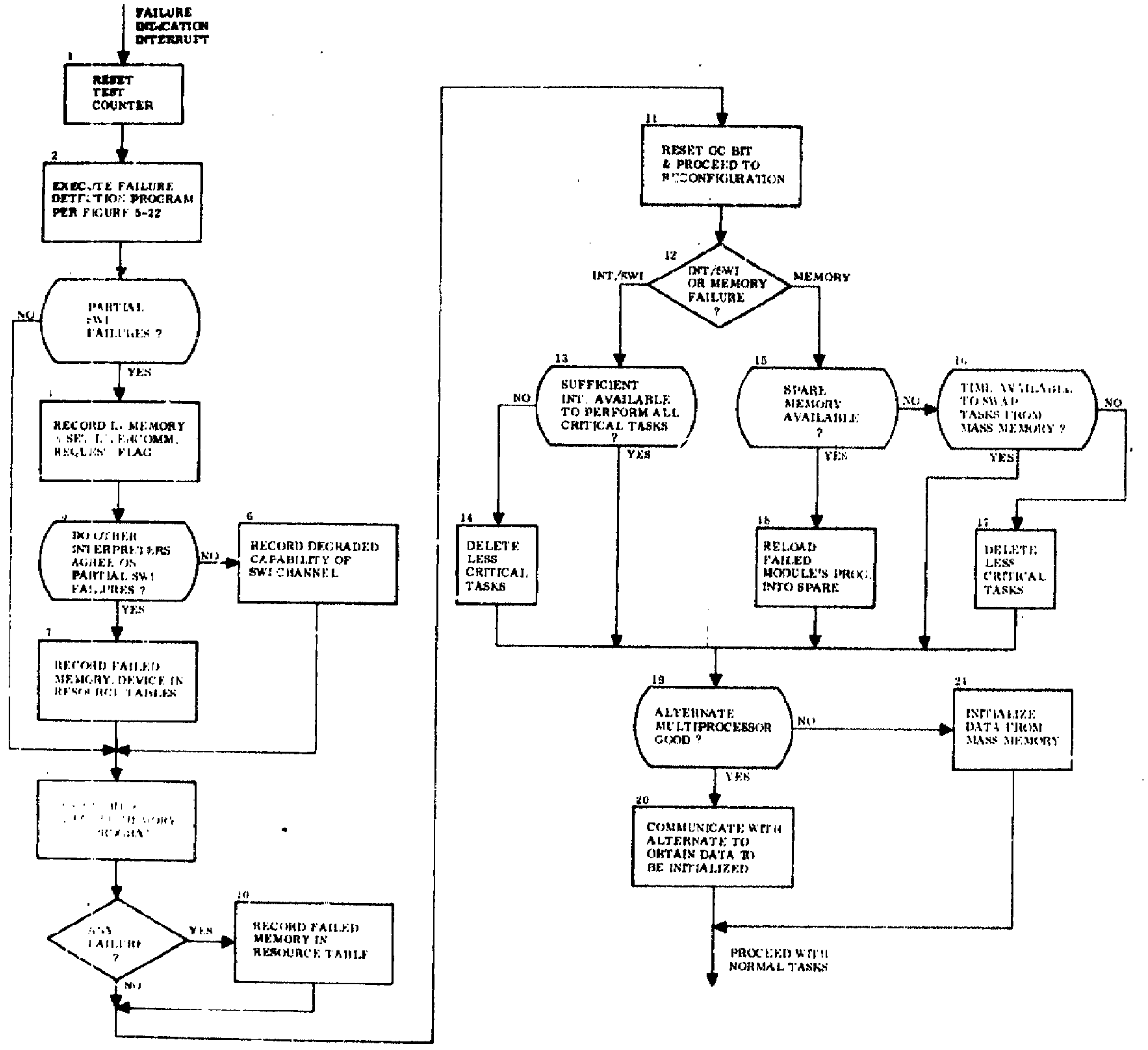


Figure 5-23. Failure Isolation/Reconfiguration Program

the EX2 condition bit. The interpreter in normal operation will test the EX1 condition bit (any interrupts) in the last microinstruction of an 'S' instruction. If the EX1 bit is true, then the interpreter will test the EX2 bit to determine if this interrupt was caused by its own test counter, if not, then the interpreter will ret) .eve the interrupt register.

The test counter is reset in the interpreter if the EX2 bit was true, so that the interpreter can access main memory and execute the failure detection program. The EX2 condition bit also causes the power up or cold start sequence to be entered. This sequence accesses location 0 of all memory modules to determine where to go next. Following this procedure prevents a failed memory from inhibiting the failure isolation program from being entered. It should be noted that if the interpreter actually failed, it would not enter the failure isolation/reconfiguration routine unless it failed in a state where it is testing the EX1 or EX2 condition bits and also is functioning so that it can process the interrupt and execute microinstructions to follow the process i · Figure 5-23.

For purposes of this analysis it is assumed that some form of memory protect is used on program/constants in main memory. This assumption allows one to proceed on the premise that such information is not altered except due to a failure of a particular memory module itself. Information that is critical to the failure detection, isolation, and reconfiguration process will be stored in two memory modules such that it cannot be destroyed except by those two memory modules failing simultaneously.

If the interpreter/SWI channel pass the failure detection program, the test counter will be reset and failure isolation will be entered in Block 3 of Figure 5-23. If any failures with particular memories/devices were detected, they will be isolated to either the SWI channel or the particular memories/devices in Blocks 4 through 7. This will be accomplished by intercommunication with the other interpreters in the multiprocessor. If the SWI channel of a particular interpreter is inoperative with selected memories/devices, the degraded capability of that interpreter will be noted in the resources tables. Such an interpreter may or may not be used depending on the sophistication of the executive used in the multiprocessor. Likewise if the failure was a memory or device this would be recorded in the resources tables.

The memories are tested by one interpreter and a program similar to that used in the normal failure detection process is used. The isolated failure is recorded and the process transfers to the reconfiguration phase as shown in Figure 5-23.

The basic tool of the failure detection process was the test counter and the basic tool in the failure isolation process described here is the power switch in the SWI channels. This switch is driven by the test counter and prevents a failed interpreter from causing an apparent failure of the complete multiprocessor. Such a failure could occur if the interpreter were making incorrect memory requests: this could degrade the performance of the system due to the failed interpreter stealing memory cycles from good interpreters or data in the memories could be destroyed by the failed interpreter making the system completely in operative.

Additional failures that could seriously effect the system arc an interpreter issuing false GC (global condition) signals. Part of this problem with the GC logic was corrected by the modification shown previously in Figure 5-21. The remainder of this problem with the GC logic is eliminated by placing the GC logic in the SWI channel where it will be disabled with power off. The same discussion applies to the INT (interrupt) bit logic, it should be placed in the SWI channel. The power switch in the SWI channel thereby allows failure isolation and subsequent reconfiguration to be successfully implemented.

5.3.6 Reconfiguration

As discussion above the reconfiguration process described in Figure 5-23 will be entered after the failure isolation process has isolated the failure and recorded the status of the multiprocessor in the resource tables. The type of failure will first be examined, as shown in Figure 5-23, to determine the exact reconfiguration process to use.

If the failure was an interpreter or SWI channel (failures are not isolated between these two modules since one is useless without the other), the task requirements would be compared with the interpreter resources available. If sufficient interpreter resources are not available to perform critical tasks (usually tasks that must have periodic timing maintained), then less critical (e.g. certain background type of tasks) tasks would be deleted. Deletion of cirtical tasks would also be carried out on a priority basis depending on the state of the interpreter resources.

If the failure were a memory module, the first step to perform is to determine if a spare memory module is available. An available spare module would be reloaded with the failed modules program from mass memory. If no spare memory modules are available, it will be necessary to determine if sufficient interpreter capability exists such that processing tasks can share main memory by being swapped out of mass memory. If this is not feasible, then less critical tasks must be deleted.

After the above reconfiguration steps have been completed, the state of the other multiprocessor (when using the concept shown in Figure 5-19) will be examined. If the other multiprocessor is operating properly, then it will be performing temporary backup functions for the multiprocessor that was doing reconfiguration. The reconfigured multiprocessor will communicate with the other multiprocessor to inform if of a successful reconfiguration and to obtain ary data from it that may be needed in reinitializing the reconfigured multiprocessor. If the other multiprocessors were not functioning properly (or in sy ems using only one multiprocessor), then the reconfigured multiprocessor would be reinitialized using startup parameters from mass memory.

The above process assumes that program and constants in main memory are protected against writing. If this is not the case, then additional steps will be required in the isolation and reconfiguration process. Some means of verifying the integrity of this information, or of reloading the entire main memory must then be provided in the multiprocessor.

5.3.7 Other Approached Considered

Alternate approaches to the failure detection, isolation, and reconfiguration process were considered. One basic approach that should be noted, is using a method for failure detection similar to that proposed by Burroughs in reference 18. This approach eliminated the need for the test counter in each interpreter. Essentially, this counter is then implemented by software. Each interpreter has a 'time due' to report in to a table. All the interpreters check on each other to make sure the interpreters report is in on time, this is functionally very similar to the test counter.

This method works about as well as the test counter for failure detection except it does not work with only one interpreter in the multiprocessor, failures cannot be detected since there is no other interpreter to check on the failed interpreter.

There are several drawbacks in implementing isolation and reconfiguration processes.

This approach also makes failure isolation difficult since it requires interpreters checking other interpreters. A failed interpreter can report good interpreters as faulty and can induce apparent failures in other interpreters. It is difficult to acheive failure isolation in the case of conflicting failure detection reports, some means of software or hardware voting may be required to carry out failure isolation. Some form of power switching will also be required in this approach. Since the failure indication signal is not autonomous from an interpreter, then it must be provided by the other non-failed interpreters in the system in order to effect power switching. This introduces some difficulty in the isolation and reconfiguration process since failed interpreters would have the capability of issuing erroneous power switching signals.

5.4 MULTIPROCESSOR EXECUTIVE

5.4.1 Introduction

Burroughs has designed an executive structure for the multiprocessor system. This software structure, described in Ref 18, is designed to accommodate a wide range of computing activities and hardware/firmware configurations. It is therefore quite general and highly flexible. In particular many of the executive functions are designed for an environment in which processing tasks with varying and unpredictable computational requirements (memory, throughput, I/O, utilities, etc.) can be asynchronously entered into or removed from the computer system. This environment is typical of a batch-oriented data processing facility.

The executive or more accurately, the operating system to support this type of processing must include features to accommodate:

1. Job (task) insertion/deletion on-line to the system

2. Dynamic allocation of system resources (particularly memory and I/O) during system operations

3. On-line debugging of tasks during system operation

4. Variation in system configurations in terms of number and type of I/O devices, memory, etc.

A real-time aerospace application such as the ASB avionics system represents a rather specific class of processing requirements which differs significantly from non-real time data processing.

The functions/objectives of the executive software are affected by the characteristics of these processing requirements. The most significant characteristics are described below:

1. Predictable Processing Load - The total processing requirements (memory, throughput, I/O, etc.) are known and fixed prior to system operation. While different combinations of processing tasks may be required for different operating modes of the system, the precise combinations are all predetermined.

2. Cyclic Processing Tasks - A high percentage of the processing tasks are cyclic - i.e., they must be executed at a predetermined frequency, such as every 1/32 of a second. Generally tasks which are not cyclic are treated as "background" and are guaranteed some maximum completion time by an analysis of the total throughput.

3. Limited User Interaction - The system "user" (the ASB crew) has a limited and rigorously defined interface (interaction with the processing software) being able only to select from predetermined various modes, options and parameters of system operation. Additional processing tasks cannot be entered into the system during operation and no software debugging is performed when the system is on-line.

Software executives for this type of application are generally highly specialized to optimize task scheduling efficiency and minimize executive overhead. This is accomplished at a sacrifice in flexibility/generality, taking advantage of the specific processing characteristics noted above.

While a highly flexible/general executive structure such as described by Burroughs could potentially accommodate the ASB processing requirements, many of the features would not be used and significant inefficiency and overhead would result. On the other hand, some executive overhead is warranted in order to make effective use of the unique characteristics of the Burroughs Multiprocessor. In designing the executive for the Burroughs Multiprocessing system for the ASB avionics system, the structures and terminology described by Burroughs in Ref 18 were retained as much as possible and the basic philosophy of that structure remained intact. The executive design is described in the following paragraphs. A significant deviation from or modifications to the Burroughs design are noted and additional detail specific to the ASB application is presented.

5.4.2 System Configuration

The executive design is based on a multiprocessor configuration as shown in Figure 5-20. The computer system consists of one or two multiprocessors and a mass memory.

A multiprocessor consists of one to five interpreters, one to eight memories, one to eight devices, and a switch interlock. Each interpreter can access (read/write) any memory module or device through a portion of the switch interlock dedicated to the interpreter. In order to access a device, an interpreter must request a "lock" to the device and when locked, no other interpreter can access the device.

Each multiprocessor has an executive. The executives are identical for all multiprocessors except for data in the System Control Segment (described in a later section) which identifies the specific hardware in the multiprocessor and the specific tasks which it is to perform.

Each interpreter has a hardware test counter which sets a failure indication signal if it is allowed to run without being reset for a fixed length of time. The test counter is reset periodically by the executive software as long as the interpreter is operating satisfactorily. The failure indication signals from all the interpreters in a given multiprocessor are "ORed" together to form a Multiprocessor Failure Indicator (MFI) which is routed to the interrupt module of the other multiprocessor (if there is one) in the system. This MFI interrupt is used to initiate on-line backup operations in the other multiprocessor. The MFI signal is also routed to the interrupt module internal to the multiprocessor to enable interpreters to initiate failure diagnosis and recovery. Run-out of the test counter causes the interpreter to transfer control to a fixed location in the microprogram memory from which the executive attempts to diagnose and recover from the failure.

The computer system contains a Real Time Clock (RTC) which generates a signal at a fixed interval (1/64 or 1/32 of a second). This signal is routed to the interrupt modules in each multiprocessor. This signal is used to schedule real-time processing.

5.4.3 Executive Structure

The executive software consists primarily of a set of microprogram modules and a data structure called the System Control Segment (SCS). As in the Burroughs design, the executive functions are distributed among the interpreters, i.e., an executive module can be executed by any interpreter whenever required. A single System Control Segment is stored in main memory for each multiprocessor. The SCS defines the sequence of processing tasks to be performed and the status of the various hardware modules of the multiprocessor. Each interpreter accesses the SCS to determine the next task to perform. The entries in the SCS are "locked" using the Global Condition bits to prevent access by one interpreter while an entry is being modified by another interpreter. In the Burroughs design, microprogram executive modules were stored in main memory (or some off-line storage) and when a particular interpreter required an executive function, it would load the appropriate module from main memory into its microprogram memory. In the ASB system all the executive modules would be permanently resident in ROM in the microprogram memories. The cost and overhead of using alterable microprogram memories and dynamic microprogram memory allocation is not justified for the ASB application since the executive's functions are small and relatively constant during system operation. The Locator and Allocator modules and the parts list described by Burroughs were used to locate and load executive modules into microprogram memory and are therefore not required in the ASB system.

5.4.3.1 System Control Segment

The term System Control Segment (SCS), borrowed from the Burroughs description refers to a set of data tables which define the status and schedule of tasks for a particular multiprocessor. The SCS is the data which directs and coordinates the executive functions in each of the interpreters. The SCS contains the following tables.

Task Table - This is a simplified, slightly modified version of the corresponding table in the Burroughs design. It is used for scheduling unscheduling processing tasks. It contains one entry for each task in the multiprocessor. An entry contains the following information:

ENTRY INACTIVE BIT	READY-TO-RUN BITS	COMMON TASK BITS	WORK AREA POINTER

1. Entry Inactive Bit: This bit is reset when the corresponding task is being processed by some interpreter.

2. Ready-to-Run Bits: These bits indicate the status of the task in terms of execution criteria. When all the bits are set, the task is "ready to run."

3. Common Task Bits: These bits are set when the task is to be executed by a specific interpreter or combination of interpreters.

4. Work Area Pointer: This field contains the main memory address of the work area for the task. (Description of work area is defined later.) In the Burroughs design, task priority was part of the information stored in the task table which requires that the entire task table be searched each time a new task is to be scheduled in order to find the high priority, ready-to-run task. Since all tasks in the ASB application are predetermined, their priority can be pre-assigned. Hence, the task table will be ordered by task priority and the next task to be scheduled in the system will always be the first entry (from the top) whose ready-to-run bits are set.

Interpreter Table - This is a simplified version of the corresponding table in the Burroughs design. The table contains an entry for each interpreter in the multiprocessor which specifies the status of the interpreter. An entry contains the following information.

Entry Inactive Bit	Interpreter Down	Task Number	Communication Area

1. Entry Inactive Bit: This bit is reset to indicate that the entry is being modified.

2. Interpreter Down: This bit is set when the interpreter is non-operative.

3. Task Number: This field contains the number of the task table entry currently being processed by the interpreter.

4. Communication Area: This portion of the entry is used for temporary storage by the interpreter.

Note that the function of the "start time," "wait time," and "time next report due," fields in the Burroughs design have been replaced by the hardware-implemented test counter which acts as a "watchdog" timer for the interpreter.

Memory Map - In the Burroughs design, main memory was segmented into 256 word pages and allocation/status was maintained at this level. In the proposed B-1 design, main memory is composed of 4 K or 8 K modules and replacement/reconfiguration is performed at the module level. Therefore, the memory map contains an entry for each module in a multiprocessor. The entry contains the following information:

FIRST ADDRESS		LAST ADDRESS		
Entry Inactive BIT	STATUS #1	STATUS #2	. . .	STATUS #n

1. First Address: The main memory address of the first word of the module.*
2. Last Address: The main memory address of the last word of the module.*
3. Entry Inactive Bit: This bit is reset to indicate that the entry is being modified.
4. Status #1: This bit is set to indicate that this module has successfully passed the test performed by the i^{th} interpreter.

Resource Table - This is a greatly simplified version of the corresponding table in the Burroughs design and contains an entry for each external input/output device or channel. Each entry contains the following information:

Entry Inactive BIT	ID	Status

1. Entry Inactive Bit: This bit is reset to indicate that the entry is being modified.
2. ID. A unique identification of the resource.
3. Status: The current status of the resource; i.e., operational/non-operational and fault indicators.

The additional information stored in the resource table in the Burroughs design was related to use of general purpose resources such as card readers, printers, tape units, etc., and is not appropriate/necessary in the ASB system.

5.4.3.2 Table Locks

Since the data tables in the system control segment are accessed by all the interpreters in the multiprocessor, a mechanism must be included to prevent conflicts in using the tables; i.e., one interpreter uses an entry while another interpreter is modifying it or vice versa. The locking philosophy described by Burroughs using the global condition bits and the entry inactive bits will adequately provide the mechanism. This philosophy requires that an entry can only be modified/used when its entry inactive bit is set. The entry inactive bit can only be set when the interpreter's global condition bit is set.

5.4.3.3 Task Work Area

There is a unique work area in main memory for each task specified in the task table. This work area has the same function as that described in the Burroughs design, namely to define the state of the task and the task's interaction (via ready-to-run bits) with other tasks. The work area contains the following information:

*These fields are required to allow for variable size memory modules

TASK NO.	RR BITS RESET MASK
Interrupt BIT	TASK ENTRY POINTER
"S" Machine State	
RR Masks for Associated Tasks	

1. Task No.: This field contains the number of the task table entry corresponding to this task.

2. RR Bits Reset Mask: This field contains a mask which is to be "ANDed" with the Ready-to-Run bits in the corresponding task table entry at the completion of the task. This has the effect of resetting specific Ready-to-Run conditions.

3. Interrupt Bit: This bit is set when the task is interrupted prior to its completion.

4. Task Entry Pointer: This field contains the main memory address of the first "S" language instruction of the task.

5. "S" Machine State: This area provides storage for the "S" machine registers in the event that the task is interrupted.

6. RR Masks for Associated Tasks: This area contains a sequence of RR bit masks and task numbers. These masks are "ORed" with the RR bits in the specified task table entries.

The task work area in the Burroughs design contained additional information such as a source table and task resource table which was used to provide access to microprogram executive modules and general purpose I/O resources. This information is not necessary in the ASB design since all executive modules are permanently stored in microprogram memory and I/O resources are accessed directly in the "S" language.

In the Burroughs design, the "S" machine registers; i.e., the accumulator, registers, program counter, etc. of the emulated machine, were located in the work area for the particular "S" level task being executed. In this manner, the "S" machine state was correctly stored with a task at the completion of each "S" instruction and no additional information needed to be saved if this task were interrupted. The primary disadvantage of this approach is the overhead (execution time) required to access and restore these registers from main memory for each "S" instruction. This overhead is compounded by the potential conflicts in main memory access between multiple interpreters. The register limitations of the interpreter design were noted in Section 4.2 of this report.

5.4.3.4 Executive Modules

As previously indicated, the executive modules are segments of microcode stored in the read-only microprogram memory of each interpreter. The following paragraphs describe each of these modules.

5.4.3.4.1 S-Language Interpreter. This module interprets and executes the processing tasks which are written in an S-language. The S-language would be similar to the machine language of a large airborne computer but would also contain instructions for performing input/output. The input/output necessary to perform the avionics system processing will be either imbedded in the S-language processing tasks or programmed as separate S- language tasks.

The S-language Interpreter is entered from the Scheduler module and it returns to the Scheduler in response to an S-language instruction executed as the last instruction of each processing task. The microcode to interpret/execute each S-language instruction includes testing of the external condition bits to detect real time clock and multiprocessor failure indicator interrupts. If an interrupt is detected, the status of the "S-machine" is saved in the work area for the current processing task together with an indication that the task has been interrupted. Intepreter control is then transferred to the Interrupt Processor module.

5.4.3.4.2 S-Level Subroutines. This module is an extension of the S-language Interpreter and consists of microcoded subroutines available directly in the S-language High-usage functions such as trigonometric subroutines would be programmed in microcode to decrease the required interpreter execution time and thereby enhance the overall system throughput.

5.4.3.4.3 Scheduler. This module accesses the task portion of the system control segment to determine the next task to be executed by the particular interpreter and to update the task table entries at the completion of a task.

The scheduler is initially entered from the Initialization module after system start-up or system reconfiguration after a failure. During normal operation, the scheduler transfers control to the S-language Interpreter to execute each processing task and regains control at the completion of the task.

The scheduler tests the external condition bits at convenient points during its execution to detect real time clock and multiprocessor failure indicator interrupts. If an interrupt is detected, interpreter control is transferred to the Interrupt Processor module. The Interrupt Processor module in turn transfers control to a specific entry point in the scheduler, once interrupt processing has been completed.

5.4.3.4.4 Interrupt Processor. This module performs the functions necessary to process the real-time clock (RTC) and multiprocessor failure indicator (MFI) interrupts. The function of these two interrupts is described in subsequent sections, but the processing performed by the Interrupt Processor involves executing specific system tasks which modify entries in the task table. The result of this processing is to add specific tasks or sequences of tasks to the current list of "ready-to-run" tasks.

The interrupts are recognized and processed by all interpreters in the multiprocessor. However, the processing necessary to schedule the appropriate system/processing tasks is performed by only one interpreter -- the first one to respond. The common interrupt (INT) which can be "broadcast" by any interpreter to all other interpreters is used to coordinate this type of "system" function. The first interpreter to recognize an RTC or MFI interrupt sets the common interrupt bit in all other interpreters and stores a code word in their communication areas (in the interpreter table) indicating that the processing has been initiated. Each interpreter responding to the RTC or MFI interrupt first tests its common interrupt bit and communication area to determine whether it is the first one to respond. All interpreters after the first simply return to their scheduler modules to select their next processing tasks. (Depending on the relative timing, the interpreter may return to the task it suspended, or proceed to a new higher priority task.)

5.4.3.4.5 Self-Test Executive. This module initiates and controls execution of the multiprocessor test procedure used during initial system start up and reconfiguration. This module gets control when a test counter runout occurs or a power-up interrupt is issued. The test procedures are described in a subsequent section.

5.4.3.4.6 Bootstrap Loader. This module is used to locate and load the system loader software from mass memory or some peripheral device. This operation is necessary during system start-up (power-up) or during reconfiguration after a failure.

5.4.4 Scheduling

5.4.4.1 General

The technique used by the executive to select and execute a task is based on the approach described in the Burroughs design. The scheduling philosophy has been simplified and made more specialized, hence more efficient, for the B-1 application.

Scheduling is performed individually by each interpreter using its scheduler module and the task table portion of the system control segment. The scheduler scans the task table from top to bottom looking for the first entry which has its ready-to-run bits set. Since the task table is ordered by priority, this task will represent the highest priority task which is ready-to-run. Since all tasks are coded in the same S-language and the S-language interpreter is permanently stored in each interpreter, no overhead is required to prepare the interpreter for executing a task and all interpreters are equally efficient at executing any sequence of tasks.

Note that, by contrast, a system where multiple S-languages are used with emulators dynamically loaded into microprogram memories, significant overhead/inefficiency results from retrieving S-language emulators from main memory. This inefficiency could be potentially overcome by forming "chains" of tasks of the same S-language in the task table as described in the Burroughs design. In this way tasks in a given S-language are effectively dedicated to a given interpreter so that the interpreter will not waste time changing emulators. However, this approach has the disadvantages that the scheduling algorithm becomes more complex (hence time and memory consuming) and some of the inherent flexibility of multiprocessing is sacrificed.

In general, the task table is organized as shown in Figure 5-24. The tasks are grouped in the task table according to their execution frequency; the higher the execution frequency the higher the priority. The highest priority tasks are the system tasks which represent executive functions such as system testing which are scheduled in response to system interrupts (RTC, NFI, and Power-up). The lowest priority tasks are the background tasks which do not have a specific real-time requirement and can be executed on a "time-available" basis.

Within each frequency group, the tasks are ordered according to their ready-to-run criteria, i.e., a task which generates data needed by a second task will precede the second task in the task table. Generally the first tasks in each frequency group are I/O tasks which input data from external subsystems.

An index or pointer is stored with the task table which indicates either the task scheduled or the highest priority task which is ready-to-run. When the scheduler searches the task table for a new task, it begins at the entry specified by the task table index and it updates the pointer when it schedules a new task. At the termination of each task, the scheduler updates the RR bits specified in the work area of the task just completed. If the RR bits which are modified cause a new task to become ready-to-run, the scheduler compares the priority of the task (equivalent to its index in the task table) with the current value of the table pointer (equivalent to the priority of the most recently scheduled task). If the priority of the new task is higher, its index is stored in the table pointer. In this manner a minimum number of task table entries are scanned to schedule each new task.

5.4.4.2 Real Time Clock Processing

The real-time clock interrupts define "the processing interval for the highest frequency tasks. The lower execution frequencies are all derived from the real-time clock frequency, i.e., 1/2, 1/4, 1/8, etc. Processing at each of the execution frequencies is initiated by the interrupt processor module in response to the RTC interrupt. The interrupt processor sets a RR bit in the system task corresponding to the RTC interrupt. The RTC system task simply maintains a binary counter from which it determines which frequency groups are to be scheduled during the next real time interval. Two groups are scheduled for each RTC interrupt, the highest frequency group corresponding to the RTC frequency and one of the lower frequency groups. If the RTC frequency were 64/sec, the scheduling sequence would be as pictured in Figure 5-25 for successive RTC interrupts.

Scheduling a given frequency group involves setting a RR bit in the first (highest priority) task in the group and in any other task whose execution is not dependent on data generated by another task in the same frequency group. All other tasks in the group will subsequently get scheduled as a result of prerequisite tasks being completed and setting their RR bits.

After processing the RTC interrupt, the interrupt processor module transfers control to the scheduler module rather than returning to the interrupted module (most likely the S-language interpreter). In this manner, processing in all interpreters will be re-initiated at the highest priority waiting task -- either the one that was interrupted or the new tasks scheduled as a result of the RTC. The first task scheduled after the RTC will of course be the RTC system task which will be executed by the first available interpreter.

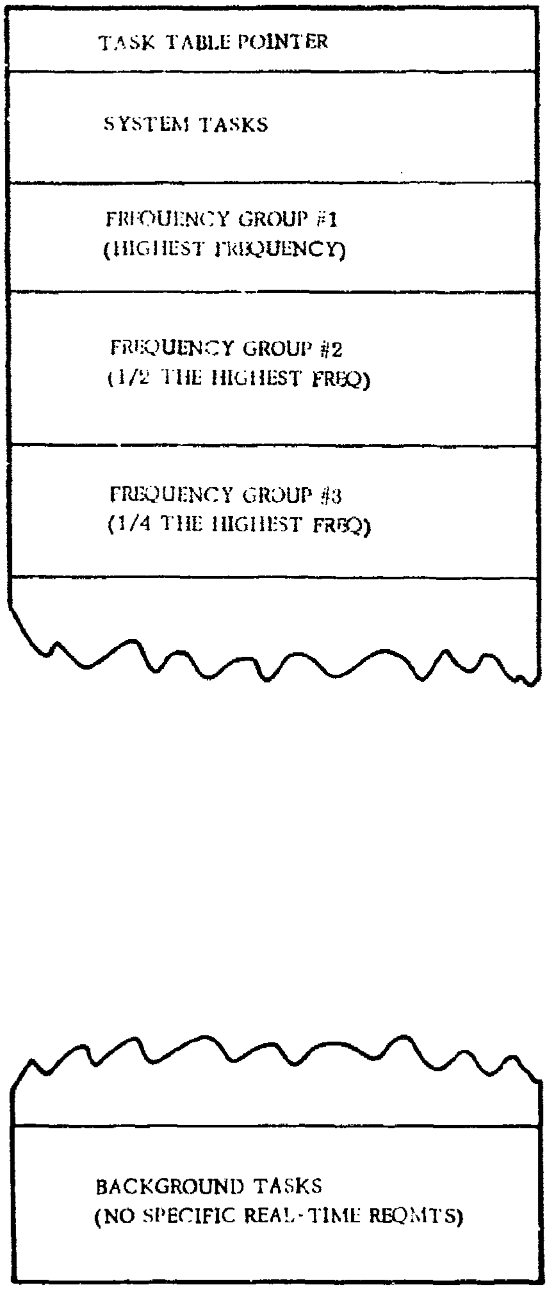


Figure 5-24. Task Table Organization

RTC 1: 64/SEC + 32/SEC

RTC 2: 64/SEC + 16/SEC

RTC 3: 64/SEC + 32/SEC

RTC 4: 64/SEC + 8/SEC

RTC 5: 64/SEC + 32/SEC

RTC 6: 64/SEC + 16/SEC

RTC 7: 64/SEC + 32/SEC

RTC 8: 64/SEC + 4/SEC

●
●
●

●
●
●

Figure 5-25. Scheduling Sequence

5.4.4.3 Task Execution

Once the scheduler locates the highest priority "waiting" task, it must set the interpreter's global condition bit in order to modify the task table entry and initiate execution of the task. The use and operation of the global condition bits has been described by Burroughs and in other portions of this report. Essentially the global condition logic provides the means by which a given interpreter can "lock out" other interpreters when necessary to avoid interpreter conflicts. Once the interpreter achieves the global condition lock, i.e., succeeds in setting its global condition bit, it again checks the "entry inactive bit" in the task table entry to make sure no other interpreter has just set it. If the entry inactive bit is still reset, the interpreter examines the "common task bits." If the entry is a normal task, i.e., one that is to be executed by only one interpreter, the interpreter sets the entry inactive bit, updates the task table pointer, and releases the global condition lock. If the entry is a task to be executed by more than one interpreter, the scheduler resets only its bit in the common task bits field, does not set the entry inactive bit and does not update the task table pointer. *

In order to initiate execution of the task, the scheduler stores the task index in the interpreter table, retrieves the work area pointer from the task table entry, and transfers control to the S-language interpreter. The S-language interpreter examines

*Note that for a "common task," the last interpreter to execute the task will set the entry inactive bit and update the task table pointer, thus removing the task from the "waiting" list.

the task work area to determine where to begin task execution. If the interrupt bit is set indicating that execution of the task had been previously interrupted, the S-machine registers (including the program counter) are loaded from the "S machine state" field of the work area. Otherwise, the program counter is loaded from the "task entry pointer" field.

Once execution of the task by the S-language interpreter has begun, the task is executed until either it is completed or it is suspended due to the occurrence of an interrupt. When a task is completed (as determined by the execution of a particular S-language instruction), the S-language interpreter returns control to the scheduler. The scheduler updates the RR bits specified in the task work area (the RR bits of the task just completed as well as those of any related tasks), resets the entry inactive bit, and proceeds to schedule the next waiting task.

If a RTC or MFI interrupt is detected during task execution, the task is suspended to allow processing of the interrupt. Suspending a task is accomplished in the S-language interpreter by setting the interrupt bit in the task work area, storing the S-machine registers in the work area, resetting the entry inactive bit in the task table, and transferring control to the interrupt processor module. The task's execution will be resumed at a subsequent time when it again becomes the highest priority ready-to-run task.

It should be noted that in a two multiprocessor system there are two MFI interrupts processed by each multiprocessor, one which comes from the "other" multiprocessor and one which is caused by a test counter runout within the multiprocessor. The former results in scheduling the on-line backup tasks while the latter initiates the multiprocessor reconfiguration procedure.

5.4.5 Failure Recovery

5.4.5.1 General

The failure detection and reconfiguration philosophy was discussed in Section 5.3. The essential characteristics of the proposed approach are summarized below.

1. Failure detection is signaled by "runout" of the test counter in one of the interpreters. Failure to reset the test counter results from either loss of program control or from a fault detected during the software/firmware self-test performed periodically by each interpreter.

2. Runout of the test counter in a given interpreter does not necessarily imply a fault in that interpreter; the fault could be with another interpreter, a memory module, or a switch interlock channel.*

3. Runout of the test counter in a given interpreter causes an interrupt to be set in the interrupt module of the corresponding multiprocessor. It also turns off the power switch in the corresponding switch interlock channel thus preventing memory or device access by the interpreter. Program control in the interpreter is forced to a fixed address in microprogram memory.

*The switch interlock is a single functional element but it is partitioned to provide an independent channel for each interpreter

4. The "ORed" result of the test counter runout (TCR) signals from all interpreters in a given multiprocessor is called the Multiprocessor Failure Indicator (MFI) and is routed to the interrupt module of the other multiprocessor (if any) in the system. This signal is used by the executive software to initiate scheduling of on-line back-up tasks.

5. When a TCR occurs, the corresponding multiprocessor stops all execution of avionic system tasks and initiates a restart procedure to isolate the fault (through comparison of results of interpreters' self-tests), reload system data, reload program data (if a memory module has failed) and reinitialize the processing tasks.

6. Reconfiguration of the multiprocessor in the event of a memory module failure involves loading (and "relocating") the programs/data into a spare module or reloading the entire system with a degraded mode configuration which requires less memory. The power switch in the corresponding switch interlock channel is used to remove a failed interpreter from the system. The system can continue to operate after interpreter failures as long as sufficient throughput is still available to perform the primary mode or degraded mode computations. Switch interlock failures are isolated on a channel basis and are not distinguished from interpreter failures.

5.4.5.2 On-Line Back Up

In a computer system containing two multiprocessors, each multiprocessor provides an on-line back-up capability for the other. In order to provide rapid switch-over and/or avoid interruption in the computation some processing tasks must be computed redundantly by both multiprocessors. In this case, use of the output data from the "secondary" multiprocessor is signaled by issuance of the MFI signal from the "primary." In other cases, back-up tasks are only scheduled upon receipt of the MFI interrupt signal.

The back-up situation is retained until the operating multiprocessor receives information that the other multiprocessor has been successfully reconfigured. This information is transmitted over the parallel channel communication path between the two multiprocessors.

5.4.5.3 Reconfiguration

Once a multiprocessor has "shutdown" as a result of a TCR indication, a reconfiguration procedure is initiated in each interpreter under control of the self-test executive module in microprogram memory. The procedure is as follows:

1. A limited interpreter self-test is executed using only the internal interpreter registers and data. If this is successful, the test counter is reset which turns on power to the switch interlock channel and allows memory/device access. If this test fails, the interpreter does not reset its test counter and is effectively removed from the system.

2. All interpreters which have passed their self-test perform a read write test on each of the memory modules and store the results of their test in fixed locations in every memory module.

3. After a fixed delay timed by the interpreter's counter, each interpreter examines the results of the tests. Based on the test results, the interpreters select the operating memories* and one interpreter uses its bootstrap loader to load the system loader which in turn loads the system control segment and the program information into the good memory modules. Note that, as described in the Burroughs design, the address of the system control segment is stored in a fixed location in every memory module to ensure that it can be "found" by any interpreter. As described for the RTC interrupt processing, the first interpreter to achieve the global condition lock is the one that performs the loading function, and the common interrupt (INT) is used to signal that the task is already being performed.

4. Once the system control segment (SCS) has been loaded, the status of the system is stored by each interpreter in the SCS and the remainder of the procedure is controlled by system tasks stored in main memory and listed in the task table. These tasks include switch interlock and device tests, S-language interpreter tests, and initialization of avionics processing tasks.

5. Once the system tasks have been completed, the reconfiguration is reported to the other multiprocessor (a function of the last system task) and the scheduler proceeds with its normal operation.

*Various algorithms are possible for analysis of the multiple test results. The simplest is probably to consider a memory module failed if it fails more than one interpreter's test and an interpreter (or SWI channel) failed if a memory module passes every interpreter's test but one.

6. PHYSICAL CHARACTERISTICS

6.1 COMPUTER DESCRIPTION

The objective of this section is to estimate the physical characteristics of the Burroughs Multiprocessor, in an avionics environment, to implement the central processor for the ASB avionics system. The total computational system was described in Figure 5-18; the objective of this section is to estimate the characteristics of the computer identified as the "central processor" in this figure. The central processor actually consists of two multiprocessors. The multiprocessors are basically the Burroughs configurations as described in Section 3 with the necessary modifications required to operate in the ASB avionics system. These modifications are for the most part described in Section 5.3.2 and shown in Figure 5-20.

The resultant central processor to be mechanized is shown in Figure 6-1. The following modifications to the present Burroughs multiprocessor design are assumed:

1. Interpreter:

   a. GC and INT logic removed.
   b. Test counter (24 bits) added that counts down to 0 and is wired to EX2, it is reset by a code in nano bits 51-54.

2. SWI:

   a. Partitioned on a channel per interpreter basis as explained in Section 4.3.5.
   b. GC logic redesign and placed in SWI, INT logic placed in SWI.
   c. Power switch added on a channel basis.

3. Devices:

   a. Interrupt module added that contains a real time clock (RTC), an interrupt register, and an interrupt mask register.
   b. Multiplex Interface Controller (MIC) added.
   c. Parallel Channel (PC) added.
   d. Mass Memory Channel (MMC) added.
   e. Discrete I/O added.

The characteristics of each multiprocessor are given in Table 6-1. Basically these characteristics provide a one to one correspondence with the Burroughs lab prototype version described in Reference 1 except that the necessary modifications listed above have been added as shown in Figure 6-1.

PRECEDING PAGE BLANK-NOT FILMED

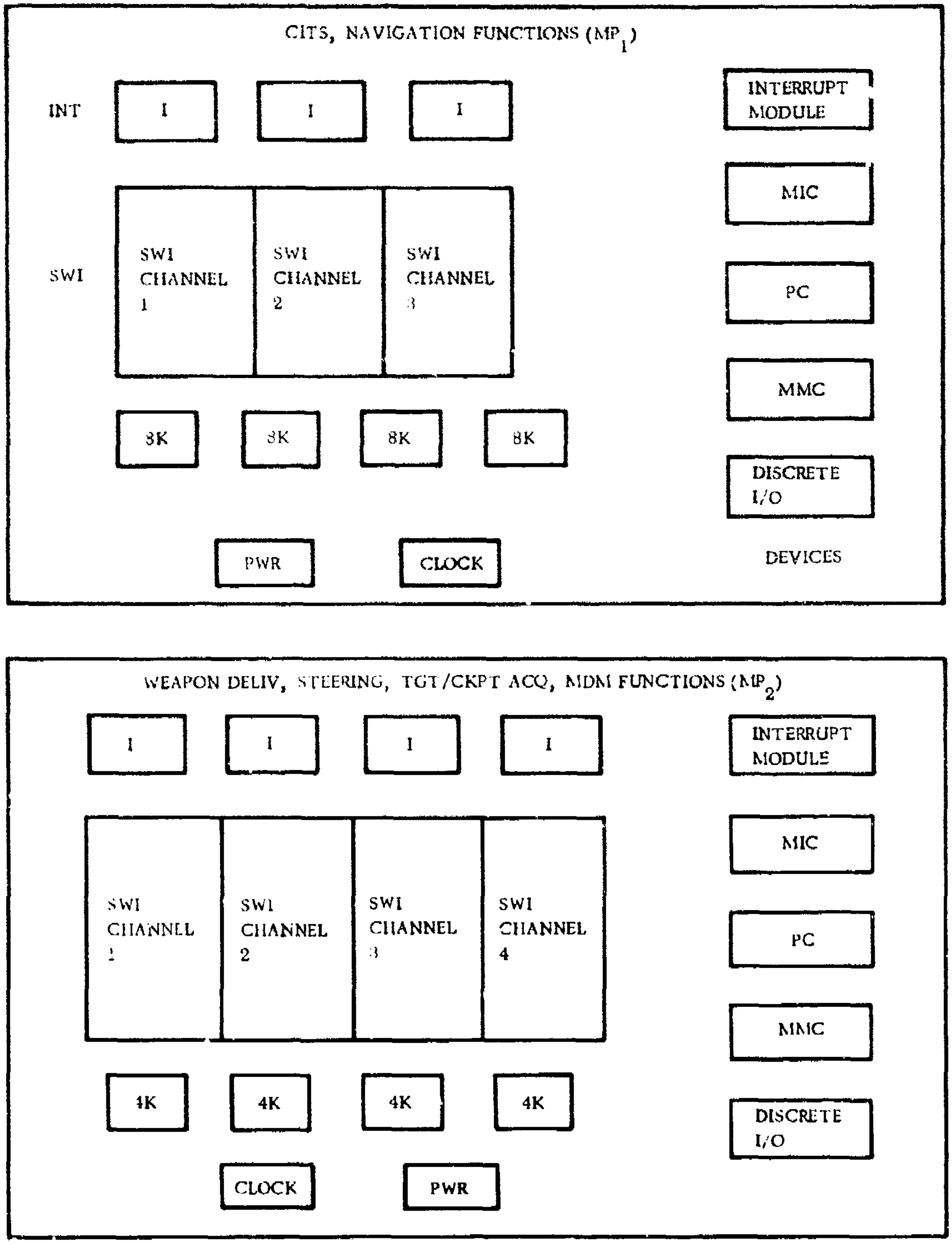


Figure 6-1. System Block Diagram of Central Processor

Table 6-1. Central Processor Characteristics

Multiprocessor 1 (MP1)

a. 3 Interpreters, 2048 x 16 bit
ROM for the MPM, 1024 x 54 bit
ROM for the NM, 4 MHz clock rate, 32 bit LU

b. 3 SWI channels, 20 MHz clock, 8 bit wide data in/out interface,
4 bit wide address interface

c. 4 - 8,192 word x 32 bit memory modules

d. Interrupt Device

e. MIC Device

f. PC Device

g. MMC Device

h. Discrete I/O Device

Multiprocessor 2 (MP2)

a. 4 Interpreters

b. 4 SWI channels

c. 4 - 4,096 x 32 bit memory modules

d. Interrupt Device

e. MIC Device

f. PC Device

g. MMC Device

h. Discrete I/O Device

The interface both within the computer and external to the computer was itemized to aid in performing the physical definition. The external interface to each multiprocessor is given below:

1. MIC - 2 twisted pair - shielded cables
2. PC - 16 signals (estimated)
3. MMC - 16 signals (estimated)
4. Discrete I/O - 8 signals (estimated)
5. Interrupt - 8 signals (estimated)
6. Memory Port - 39 signals (estimated), allows connection to main memories via a SWI channel for off line operations such as memory load.
7. Miscellaneous external control - 10 signals (estimated)
8. Power - 115 volt - 400 cycle

The internal interface is shown in Figures 6-2 through 6-4 and is summarized below:

1. Interpreter:

   a. SWI (MDC) - 13 signals, Int. Clock, and H.S. Clock.
   b. SWI (MC/DC) - 3 signals.
   c. SWI (IOSN - 4 bits data in and out, 2 bits address, and clocks) - 10 signals.
   d. SWI (IOSN - total for 8 bits data and 4 bits address for 8 memories and 8 devices) - 20 signals.
   e. Interrupt device - 2 signals.

2. SWI:

   a. MDC

      (1) Interpreter - 13 signals, Int. Clock, and H.S. Clock
      (2) Other MDC channels - 5 signals
      (3) MC/DC - 9 signals
      (4) IOSN - 1 signal
      (5) H.S. Clock
      (6) External clear signal

   b. MC/DC

      (1) Interpreter - 3 signals
      (2) MDC - 9 signals
      (3) IOSN - 5 lines
      (4) Other MC/DC channels - 32 signals
      (5) Memories - 8 signals
      (3) External clear signal

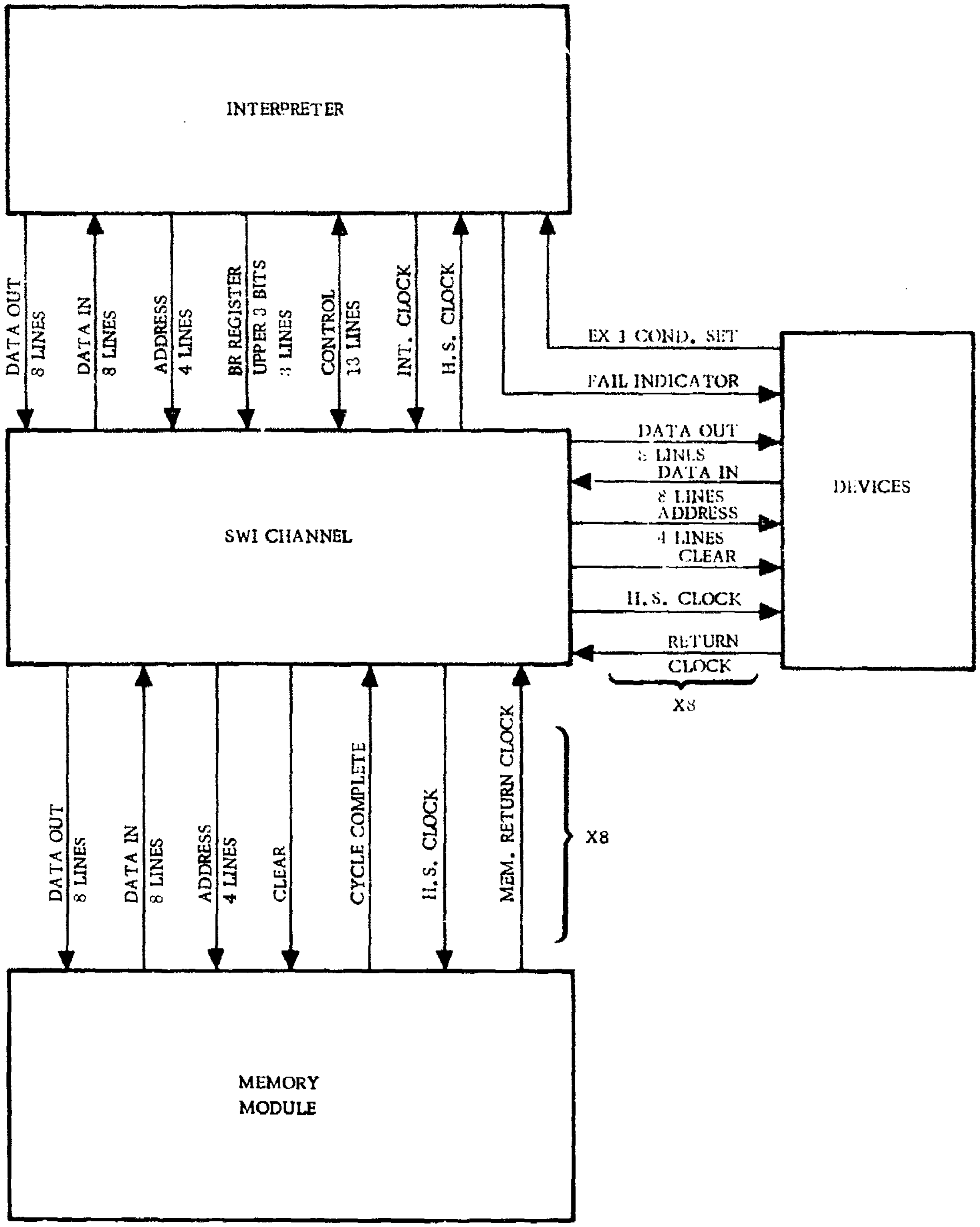


Figure 6-2. Interface Between Modules Per Interpreter/SWI Channel

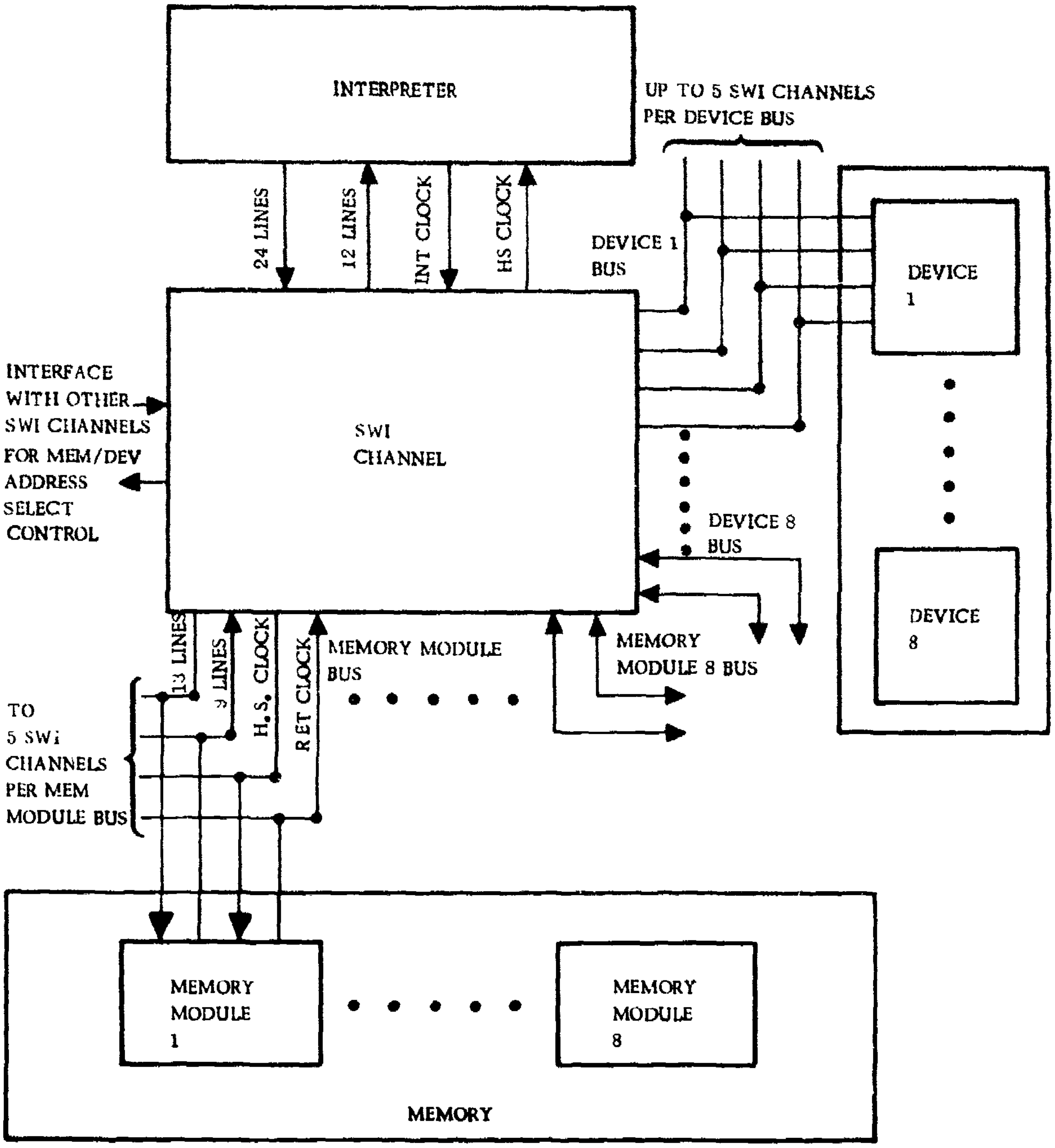


Figure 6-3. SWI Channel Interface

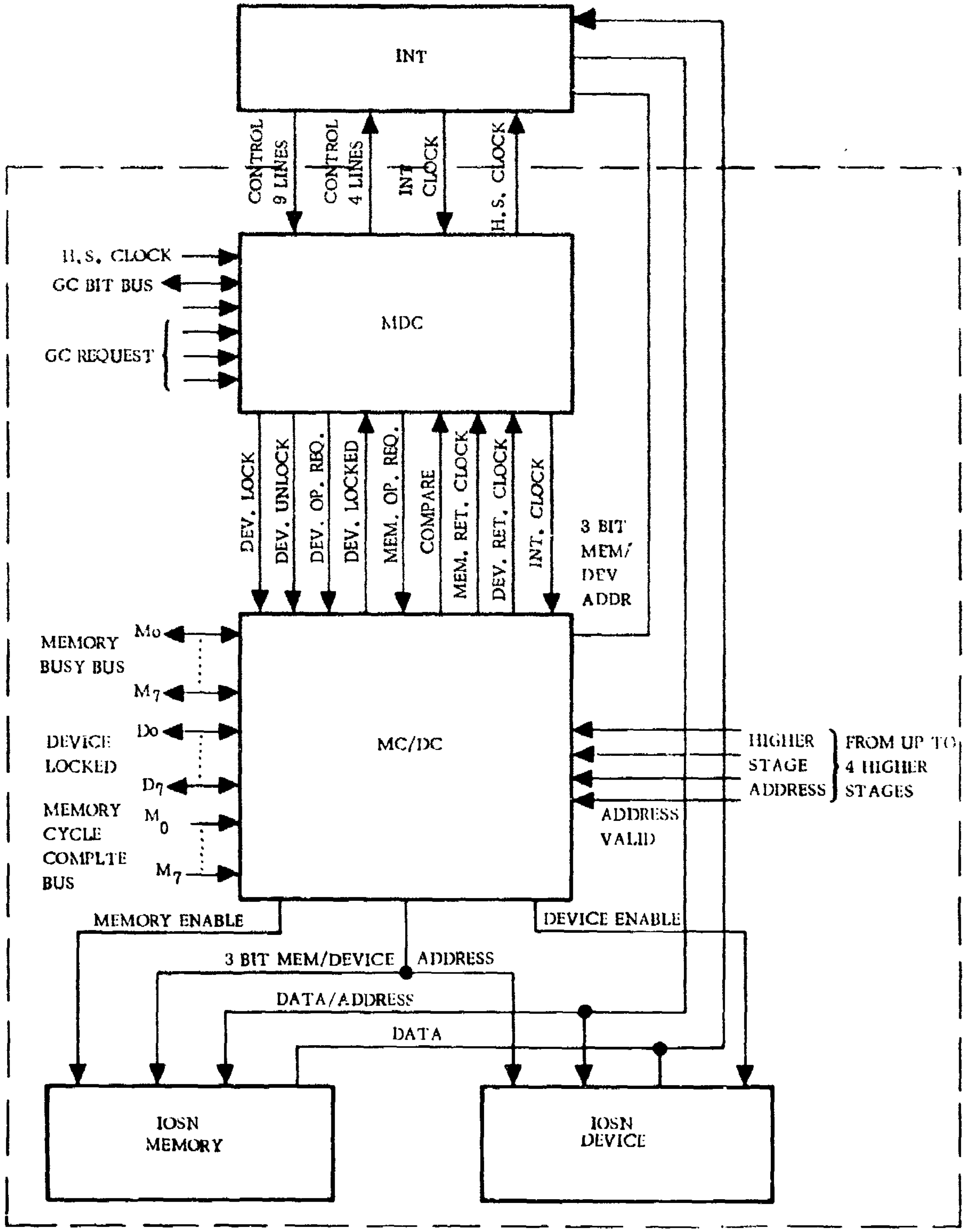


Figure 6-4. SWI Channel Internal Interface

c. IOSN (4 bits data in/out, 2 bits address)

(1) Each interpreter - 10 signals
(2) Memories or devices - 96 signals
(3) Each MC/DC - 5 signals
(4) Each MDC - 1 signal

3. Memory

a. SWI IOSN - 23 signals
b. SWI MC/DC - 1 signal

4. Device

a. SWI IOSN - 23 signals

6.2 MODULE MECHANIZATION

6.2.1 Introduction

The logic technology utilized in estimating the physical characteristics was bipolar MSI. Currently available state-of-the-art devices were used in estimating the parts count. The memory technology utilized was 2 mil plated wire which is considered a state-of-the-art technology. The packaging philosophy utilizes standard avionics practice with multilayer circuit and interconnect boards with forced air cooling. The resultant physical estimate for the central processor represents a computer mechanized from state-of-the-art technology that is readily producible requiring no technology developments or "breakthroughs."

6.2.2 Interpreter Mechanization

The basic module was a plug-in board. This module was based on approximately a 6.5 inch high by 9.0 inch wide board. A preliminary logic design of the interpreter was performed to arrive at an estimate of the number of integrated circuits required. Table 6-2 gives the integrated circuit types and amounts used in the preliminary design. Table 6-3 gives a summary of the mechanization. Allowing approximately a 20 percent spare factor for miscellaneous functions overlooked in the preliminary design, 300 IC's dissipating 25 watts are required.

Figure 6-5 through 6-8 show preliminary logic diagrams for part of the interpreter. These logic diagrams and parts counts should be considered preliminary estimates only. A final design would require detailed study of timing and control signals.

For a module of 6.5 x 9.0, or 58.5 square inches of mounting area, a considerable number of components can be mounted on a board; normal practice would allow in the neighborhood of 300-14 lead flat pack integrated circuits. However, in view of the number of interconnections internal to the module and externally into the system, the module would become very cumbersome to layout and require a substantial number of layers of circuitry to effect all the interconnections (the large power requirements not withstanding). Therefore, it is estimated that the interpreter will require two

Table 6-2. Interpreter Parts Estimate

Quantity	Device Number	Device Name	No. of Pins	Power Dissipation Typ MW	Total Power Dissipation
10	MM6255	1024 x 10 bit ROM	24	500	5000
1	74LS00	Quad 2-in. NAND	14	8	8
3	74LS04	Hex Inverter	14	12	36
2	7408	Quad 2-in. AND	14	80	160
1	7411	Triple 3-in. AND	14	40	40
1	74LS20	Dual 4-in. NAND	14	4	4
1	7423	Dual 4-in. NOR	16	45	45
3	7432	Quad 2-in. OR	14	95	285
1	74LS51	Dual 2-wide AOI	14	28	28
1	74LS55	2-wide 4-in. AOI	14	28	28
1	7460	Dual 4-in. Expander	14	8	8
5	74LS73	Dual J-K FF	14	20	100
1	7486	Quad 2-in. XOR	14	150	150
25	74LS95A	4-bit Shift Register	14	50	1250
2	74LS138	3 to 8 Line Decoder	16	32	64
1	74150	16-bit MUX	24	200	200
58	74LS153	Dual 4-bit MUX	16	31	1798
1	74154	4 to 16 Line Decoder	24	170	170
24	74157	Quad 2-bit MUX	16	150	3600
32	74LS174	Hex D FF	16	66	2112
12	74LS175	Quad D FF	16	44	528
8	74LS181	Arith Logic Unit	24	105	840
3	74182	Look-ahead Carry Unit	16	180	540
5	74LS197	Binary Counter	14	60	300
4	74LS295	4-bit SR w/3-State	14	60	240
8	8243	8-bit Position Scaler	24	315	2520
12	9006	Dual 4-in. Expander	14	7	84
32	9008	Expandable 4-wide AOI	14	40	1280
					21,418 MW
					≈ 21.5 watts

Table 6-3. Interpreter Mechanization Summary

Parts Estimate		Power
Type	Quantity	
14 pin	98	
16 pin	132	
24 pin	28	
Total	258	21.5 watts
With approximately 20 percent pad factor		
Total	300	25 watts

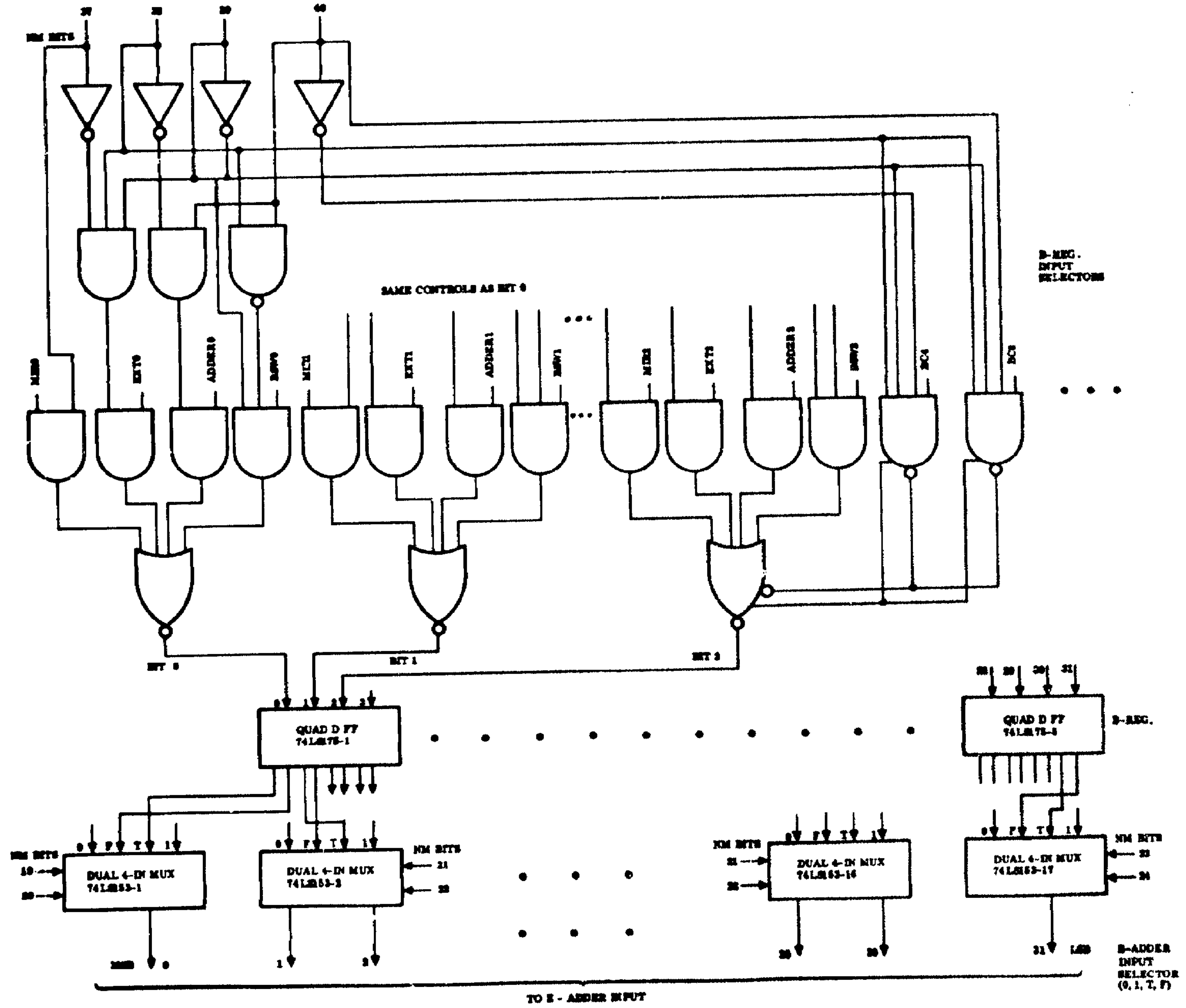


Figure 6-5. B-Register and B-Adder Input Selectors

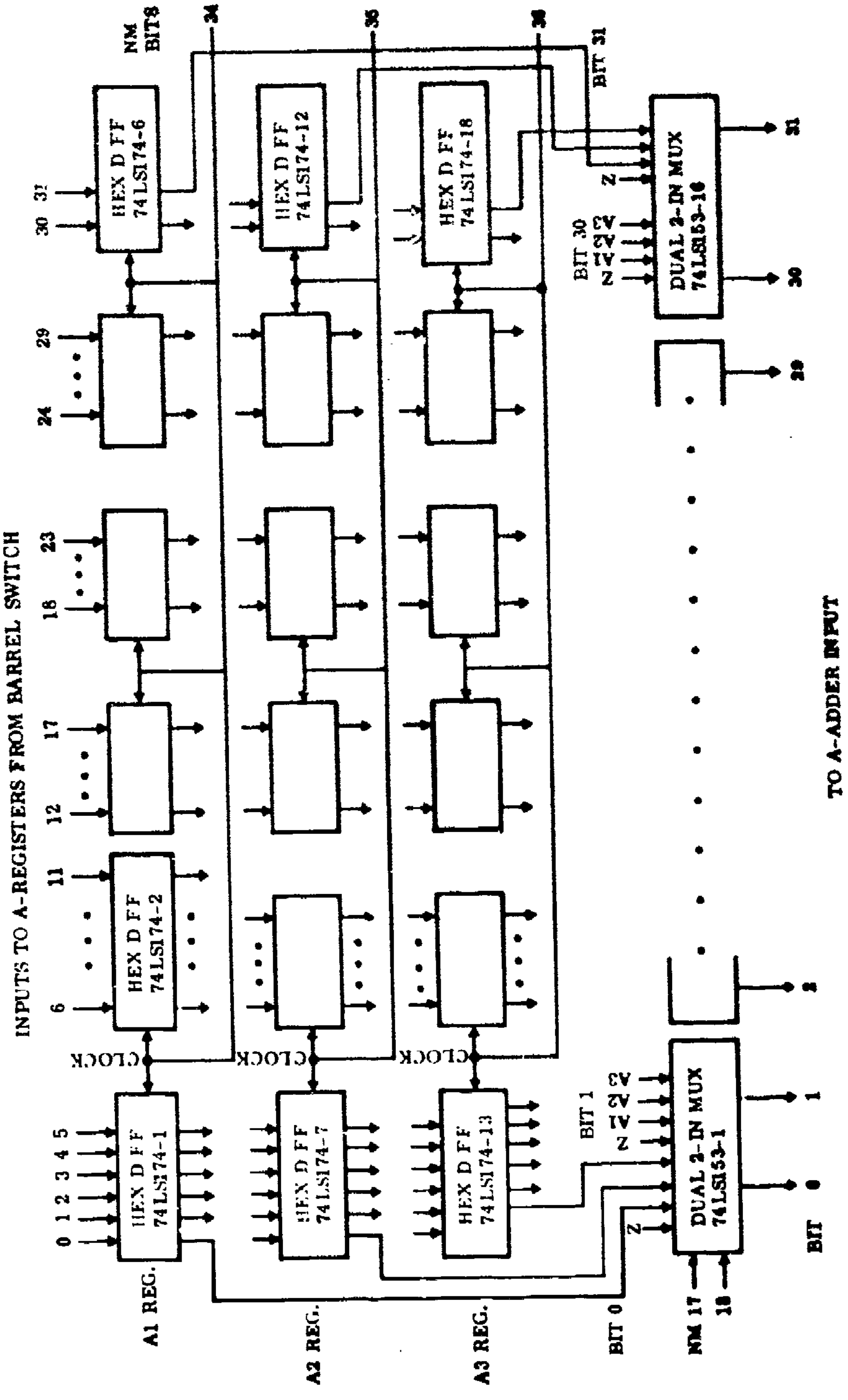


Figure 6-6. A-Register and A-Adder Inputs

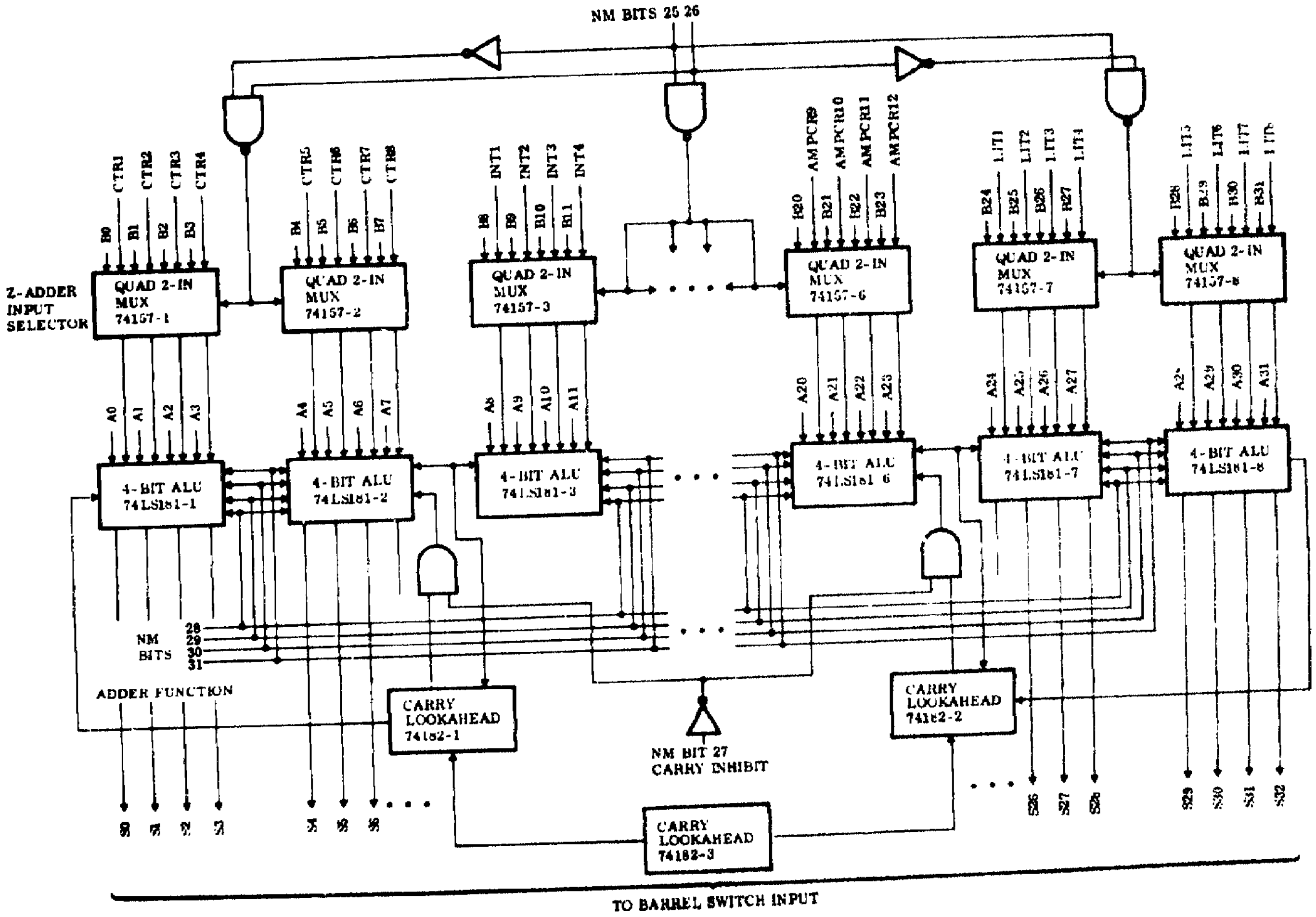


Figure 6-7. Z-Input and Adder

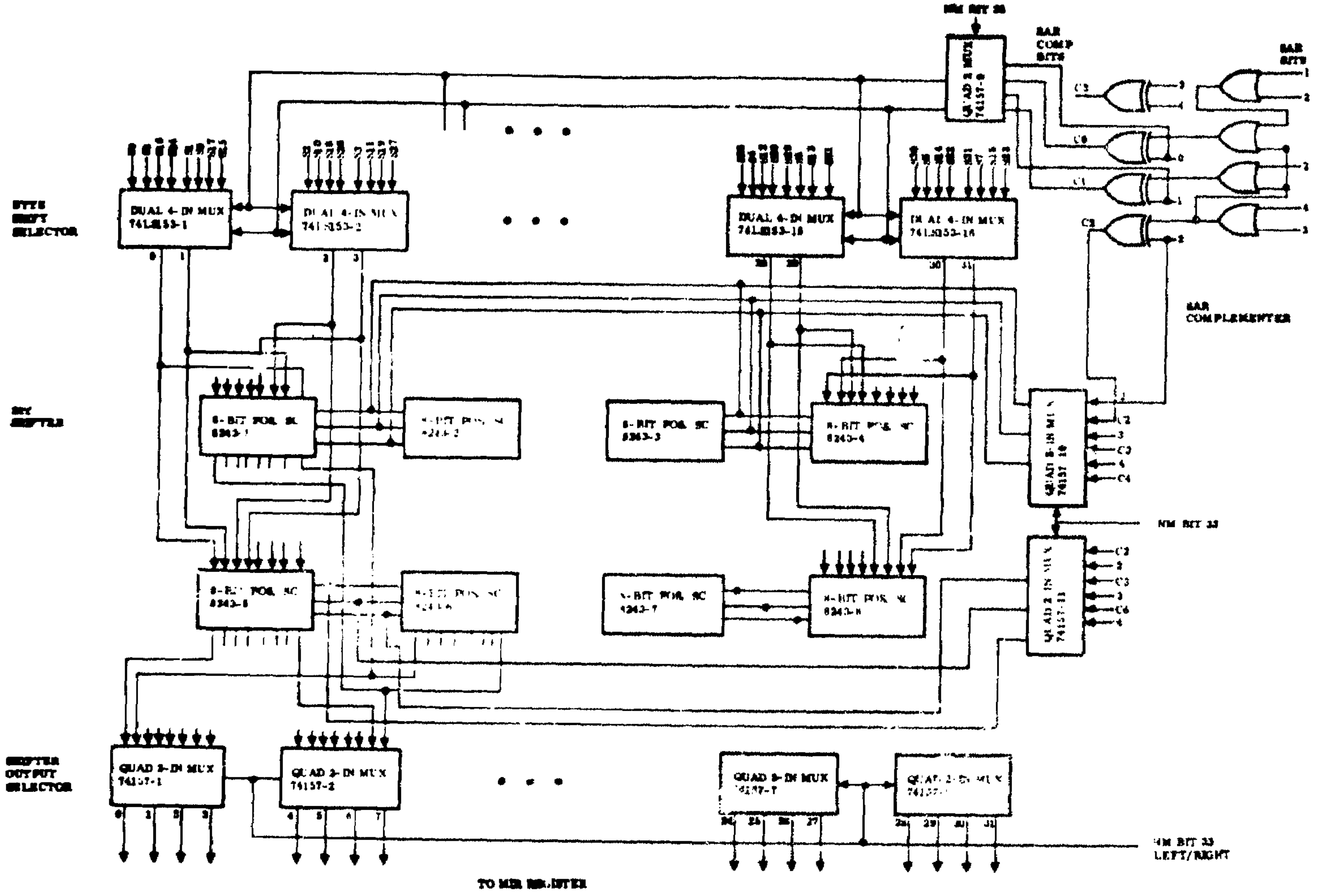


Figure 6-8. SAR Complementer and Barrell Switch

modules in its final mechanization. These two modules will be referred to as Interpreter Module A and Interpreter Module B. The exact partitioning between these two modules was not performed in this study. However a most likely partitioning will be having the MPM, NM, and MCU on one module and the CU and LU(s) on the other module.

6.2.3 SWI Mechanization

The SWI alternate design presented in Section 4.3.5 was examined to determine its method of implementation in an avionics environment. It was found that considerably higher density could be used than in the approach taken in Section 4.3.5. The resultant SWI was mechanized with two types of modules: (1) MDC-MC/DC and (2) IOSN. The MDC as described in Section 4.3.5 requires approximately 30 IC's and the MC/DC requires approximately 40 IC's, further these are primarily 14 pin IC's. It is definitely feasible to place the MDC and MC/DC on one module.

The IOSN was investigated and it was found that up to four channels (interpreters) could be placed on one module (total of approximately 55 IC's required). This IOSN provides 4 data bits in, 4 data bits out and 2 address bits for either 8 memories or 8 devices.

Therefore, the SWI was mechanized with two types of modules, one module provides all the control/selection/priority logic for one interpreter and the other module provides the required interconnection logic between memories, devices and the interpreters. It is estimated that using low power shottky devices, the MDC-MC/DC module would require approximately 3 watts and the IOSN module approximately 2 watts.

6.2.4 Memory Mechanization

The memory modules are mechanized using 2-mil plated wire technology. This technology has been pioneered by Autonetics and is now considered a state-of-the-art producible technology. The basic memory modules are constructed from a 6.585 x 9 x 0.52" board (module) that contains the plated wire array for 8K x 16 bits. This board also contains the x, y switches, diode substrates, bit axis substrates and strobe electronics. This organization allows the sense electronics to be practically integral to the memory array, thereby eliminating signal problems resulting from lengthy interconnections. Two of these boards are used to make up a 8K x 32 bit memory. In addition to the memory array boards, two electronics boards are required for the bit drivers, timing and control, address decode, data register, etc electronics. These modules will be referred to as "Memory Electronics Module A" and "Memory Electronics Module B."

Some features of the Autonetics design are listed below:

1. Small Diameter (2-Mil) Wire

   Provides minimum array size and power, and permits small, low power electronics. The wire has wide operating margins, and high output signal characteristics.

2. High Density Array

Double mat approach in conjunction with high bit density permits complete subsystem (8 K x 16 bits) on a single board, thereby eliminating the requirement for less reliable flexible cabling to other boards. Design significantly reduces interconnections and eliminates plated through holes in the mat.

3. Hybrid Packages

Repetitive circuits are combined into hybrid packages for smaller volume and increased reliability.

4. Modularized Array Configuration

Low volume hybrid packages plus high density array permit the placement of all first level electronics on the same board. This yields maximum signal-to-noise ratio.

5. Conservative Electrical Design

Two active crossovers per bit are used for balance and maximum signal margin.

Word circuit design requires relatively few parts and provides tightly controlled word current.

Bit axis electronic circuitry is greatly simplified through the use of an integrated MOS multiplexing circuit.

The plated wire plane uses Autonetics 2.0 mil plated wire and high density mat technologies to provide memory storage with low power drive requirements and high signal-to-noise output characteristics.

The planes are located on the array modules, which they share with the word and bit electronics which constitute the first level of interface with the plane. Due to a unique stacking of planes on the array module's support board, the storage capacity is twice that normally associated with a board this size. Each array board contains four plated wire mats so that a complete sense loop is accomplished on each side of the board. Figure 6-9 shows these plated wire hairpins in a cross section view. The word straps, viewed in cross section in Figure 6-10, actually wrap around the support board in addition to wrapping around each tunnel structure layer. Thus, the number of word matrix circuits is reduced from four to two since one set of circuits services the inner mats on both sides, and another set services the outer mats on both sides. Further, the number of interconnections is reduced because of the wrap around features, and plated through holes are eliminated.

The memory is organized with two crossovers per bit, which is accomplished by jumpering together adjacent pairs of hairpins where indicated in Figure 6-9. The other end of the adjacent hairpins lie on pads at the edge of the bit axis substrate, which couples them to the input of the sense multiplexer function which is on the Memory Electronics Module.

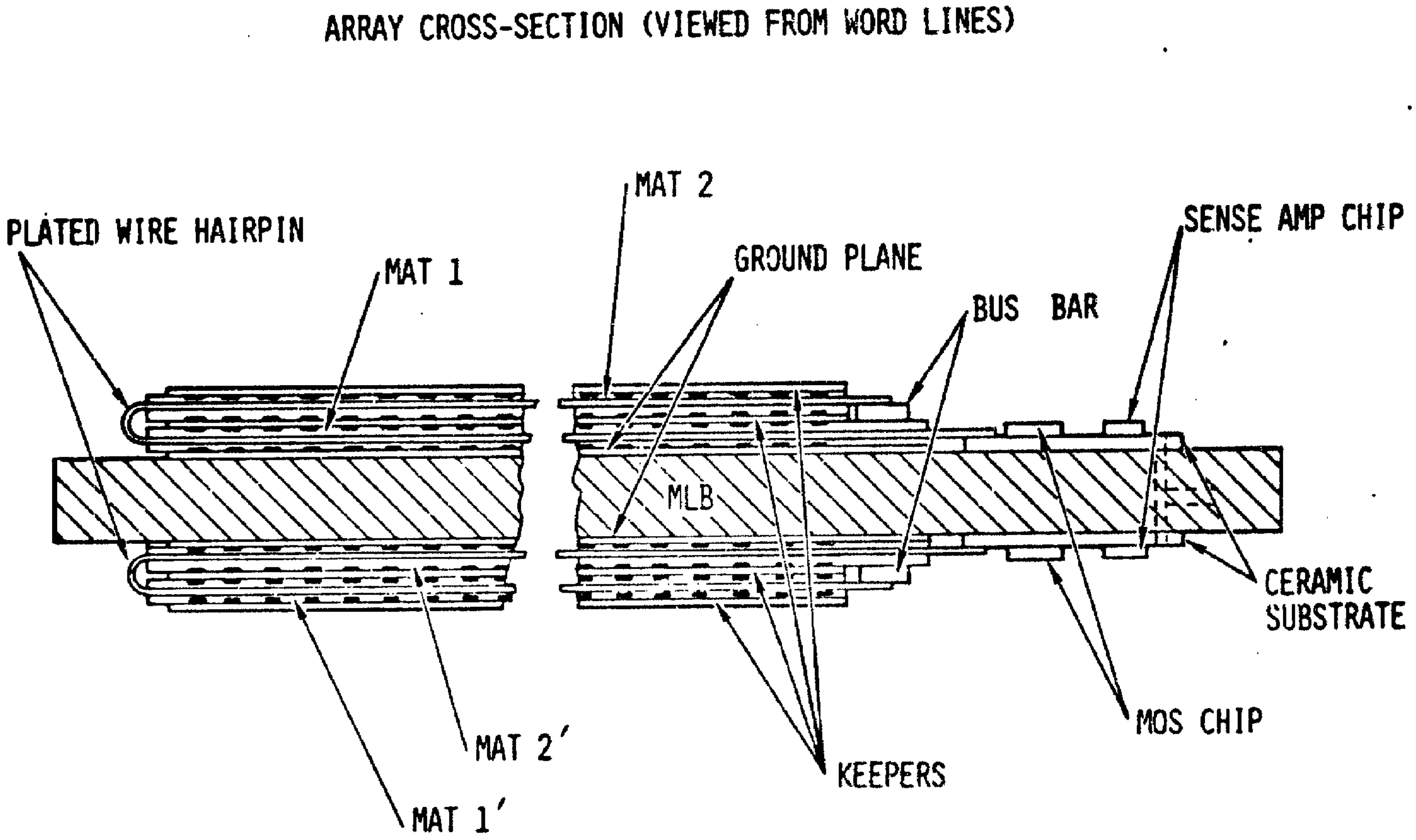


Figure 6-9. Array Cross Section (Plated Wire Plane)

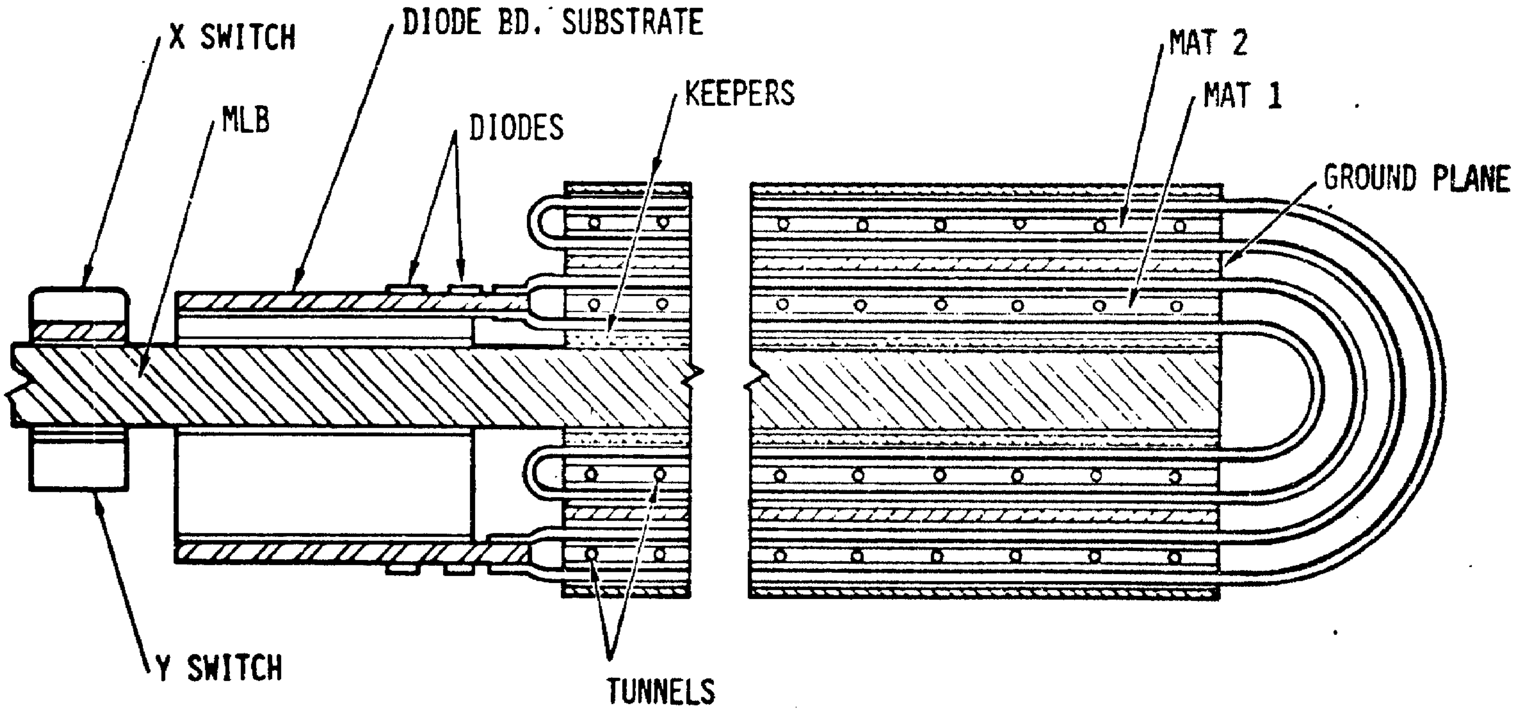


Figure 6-10. Array Cross Section (Word Strap Plane)

The word lines are composed of a one turn strap, and are joined to ceramic substrate boards containing the matrix diodes and the X and Y buses.

Keepers and ground planes are used to provide the desired magnetic field for optimum bit operation and to give proper word circuit operation.

The resulting characteristics of this plane design permit clean drive current waveshapes and high signal-to-noise output voltages. This translates into more reliable memory system operation.

The memory timing waveforms are shown in Figure 6-11. This results in the following memory operation times:

Read access:	350 nano sec
Read cycle:	800 nano sec
Write cycle:	800 nano sec

The central processor requires two types of memory modules. MP1 requires 8 K x 32 bit modules and MP2 requires 4 K x 32 bit modules. As explained above a 8 K x 32 bit module would be mechanized from two array boards and two electronics boards. Alternative configurations were examined for the 4 K x 32 bit module and it was determined that this module also would require the same basic structure as for the 8 K x 32 bit module. It is not possible to place the 4 K x 32 bit memory array on one board. Consequently the 4 K x 32 bit memory module will also require 4 boards. However, the memory array boards can have some of the components and arrays left off the boards resulting in 4 K x 16 bit array modules.

6.2.5 Device Modules

The device module types were listed in Section 6.1. The design of the MIC device was presented in Section 5.2.3.2. It is expected that this device module could be easily implemented on one board. The remaining device modules were not investigated in detail. However, it should be noted that the remaining four device modules are very simple in their mechanization and it is expected that they could all be placed on one board.

6.3 CENTRAL PROCESSOR PACKAGING

The central processor of the avionics processor and controller is packaged in two separate forced air cooled packages, one for each multiprocessor as shown in Figure 6-12. One houses the CITS and navigations functions (MP1), and the other houses the weapon delivery, steering, target/checkpoint acquisition and mission data management functions (MP2). Each multiprocessor unit is patterned after the MIL-C-172 case size of the MS 91403-C1D. It measures 7.63 inches high by 15.38 inches wide by 20.80 inches long for a volume of approximately 1.41 cubic feet. Both units weigh approximately the same, 79 pounds each. Though there are less modules in MP1, the weight difference is picked up in the larger memories. Table 6-4 lists the major assemblies for each of the units and Figure 6-13 shows the general arrangement in each unit.

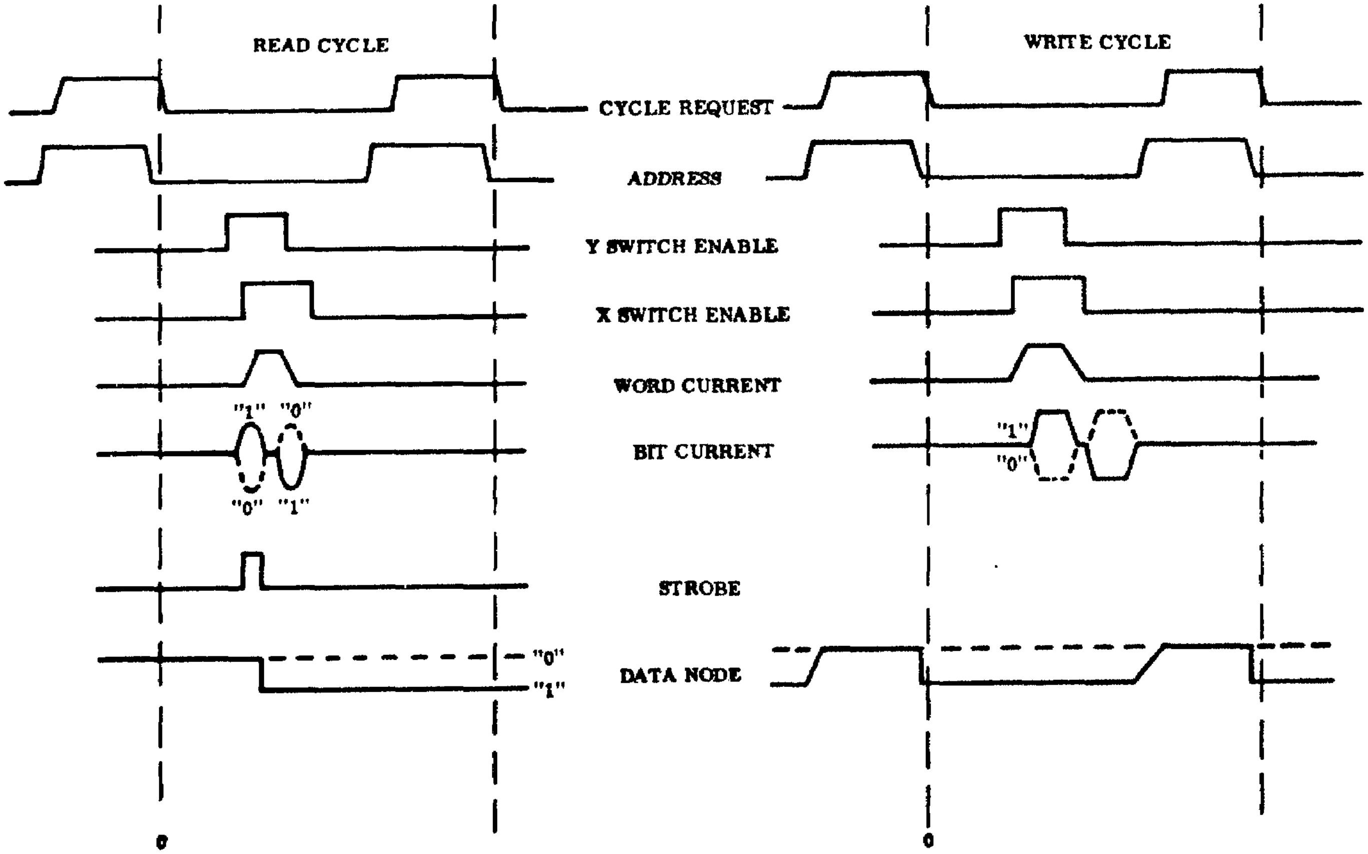


Figure 6-11. Plated Wire Memory Timing

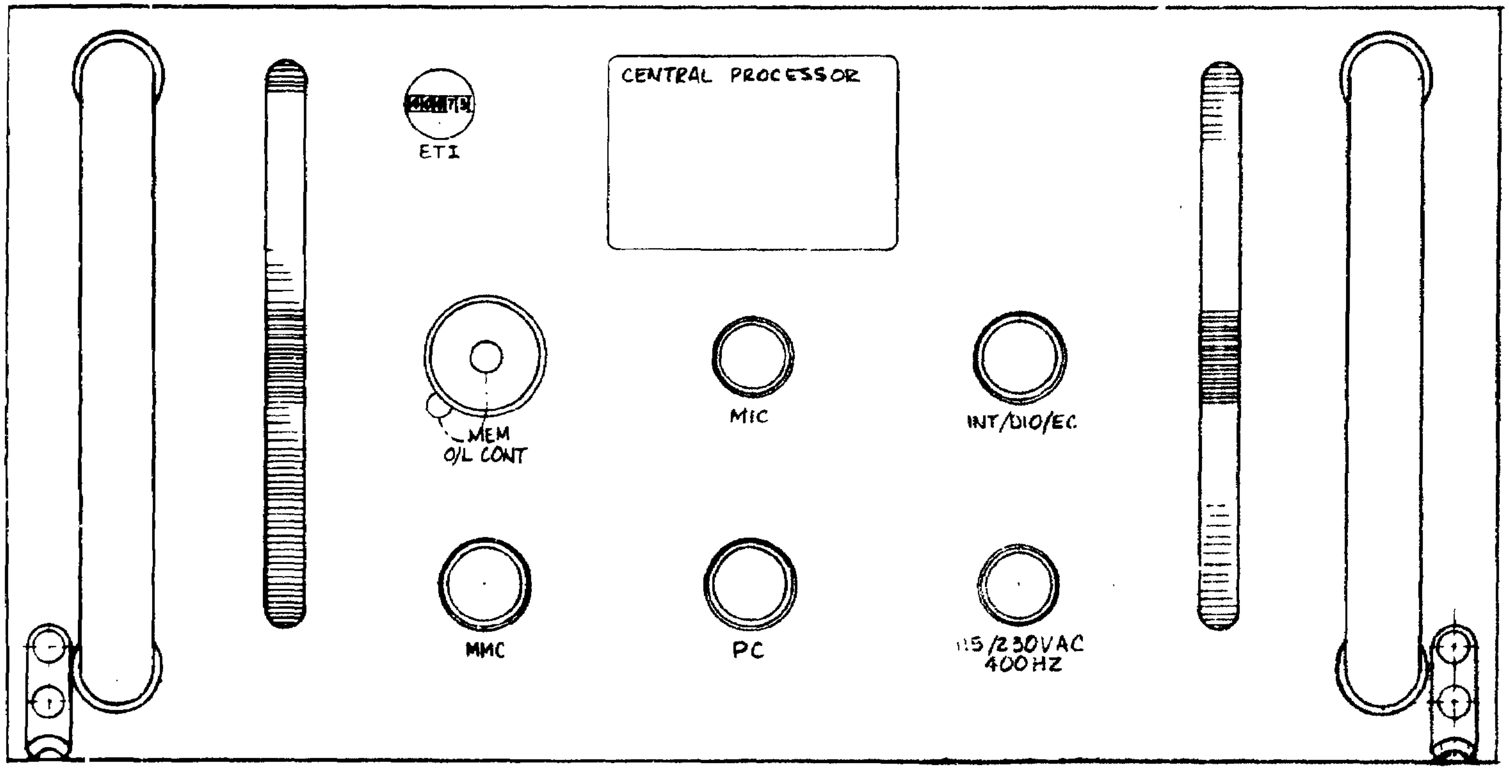


Figure 6-12. Avionics Processor and Controller

Table 6-4. List of Major Assemblies in the Central Processor

Assembly Name	MP1 Quantity	MP2 Quantity
Device Modules:		
PC, MMC, Disc I/O, INT Module	1	1
MIC Module	1	1
SWI Modules:		
IOSN Module	4	4
MDC-MC/DC Module	3	4
Interpreter Modules:		
Interpreter Module A	3	4
Interpreter Module B	3	4
Memory Modules:		
8 K x 16-bit Array Module	8	-
4 K x 16-bit Array Module	-	8
Memory Electronics Module A	4	4
Memory Electronics Module B	4	4
Power:		
Power Converter A	1	1
Power Converter B	1	1
General:		
Master Interconnect Board	1	1
Structures (Chassis, Covers, etc)	1	1

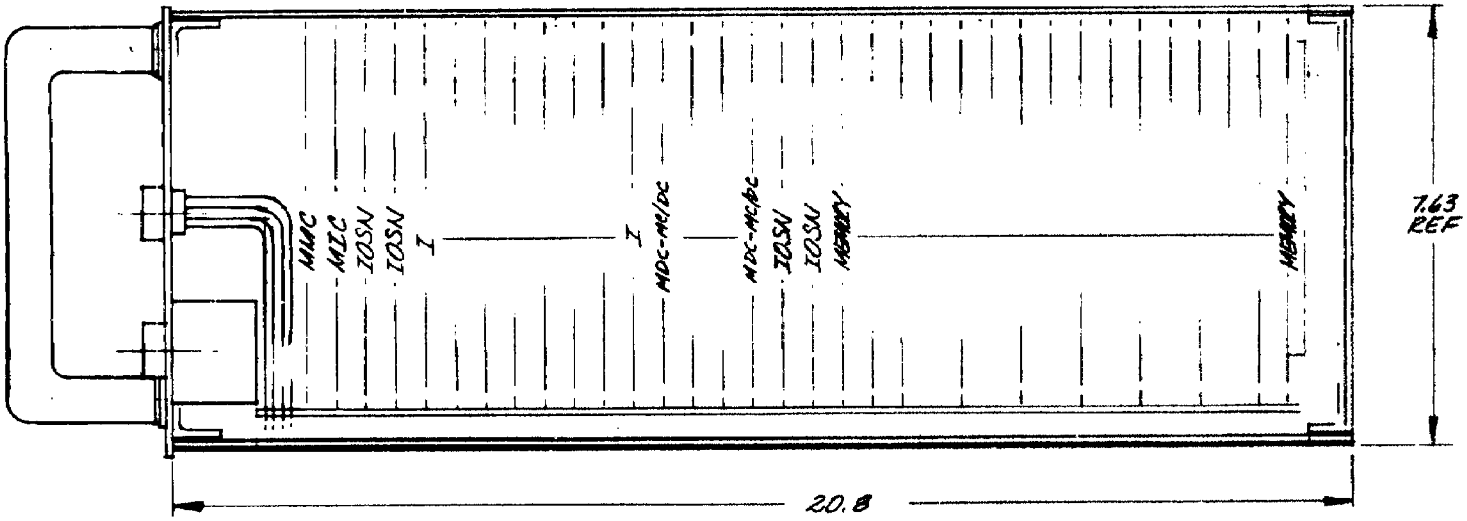


Figure 6-13. General Arrangement Avionics Processor and Controller

The housing for the units are identical in size and configuration for each of the two units. For a difference of 1.05 inches in depth to accommodate the three modules, it was decided to have one common chassis. The structure is of all aluminum construction and uses integral heat exchangers. The heat exchangers are double pinned sections serving two compartments, the power converters on the one side and the plug-in electronics and memory modules on the other. Details are shown in Figures 6-14 and 6-15. Construction techniques to be used in the fabrication of the chassis are brazing and machining with all aluminum alloys to be used in the construction.

The plug-in modules of the system are sized from a standard that is compatible with the plated wire memory modules. That is, the modules are approximately 6.5 inches high by 9.0 inches wide. In analyzing the functional requirements and in implementing the mechanization of those functions, the modules have been simplified in the layers required and the quantity of parts per module without sacrificing efficiency and functional performance requirements of the system. The most complicated module in the multiprocessor will be one of the interpreter modules (~10 layers). The remaining modules are relatively simple requiring 2-4 layer boards. Each of the modules utilizes plug type connectors to comply with the pin and socket connector arrangement required of airborne electronic equipment within specification MIL-E-5400. Heat dissipation "rails" are bonded to the boards which have in turn components bonded to them to conduct the heat out on these "rails" to the forced air cooled heat exchangers. Module locks on the principle of a wedging action are used to retain as well as help in the heat transfer from module to heat exchanger.

The power converters are in a different situation with RFI being the most problematical. The packaging for the power converter has them enclosed in "RFI tight" cans and mounted directly to the heat exchangers in "coldplate mount" fashion.

Interconnections from module to module are accomplished with a master interconnect board. Maximum number of layers to accomplish the interconnection would run approximately six layers.

The following is an estimate of the power dissipation in each multiprocessor:

Multiprocessor 1:

3 Interpreters	75 watts
SWI	17 watts
2 Device Modules	20 watts
4 Memory Modules*	65 watts
Total	177 watts
with 2/3 efficiency in Power Supply:	265 watts

*Note: The power dissipation of a memory module is approximately 20 watts when operating and 5 watts in standby. This power is approximately the same for the 4 K and 8 K memory modules. In MP1, since there is one more memory module than interpreters, one module will on the average be in standby.

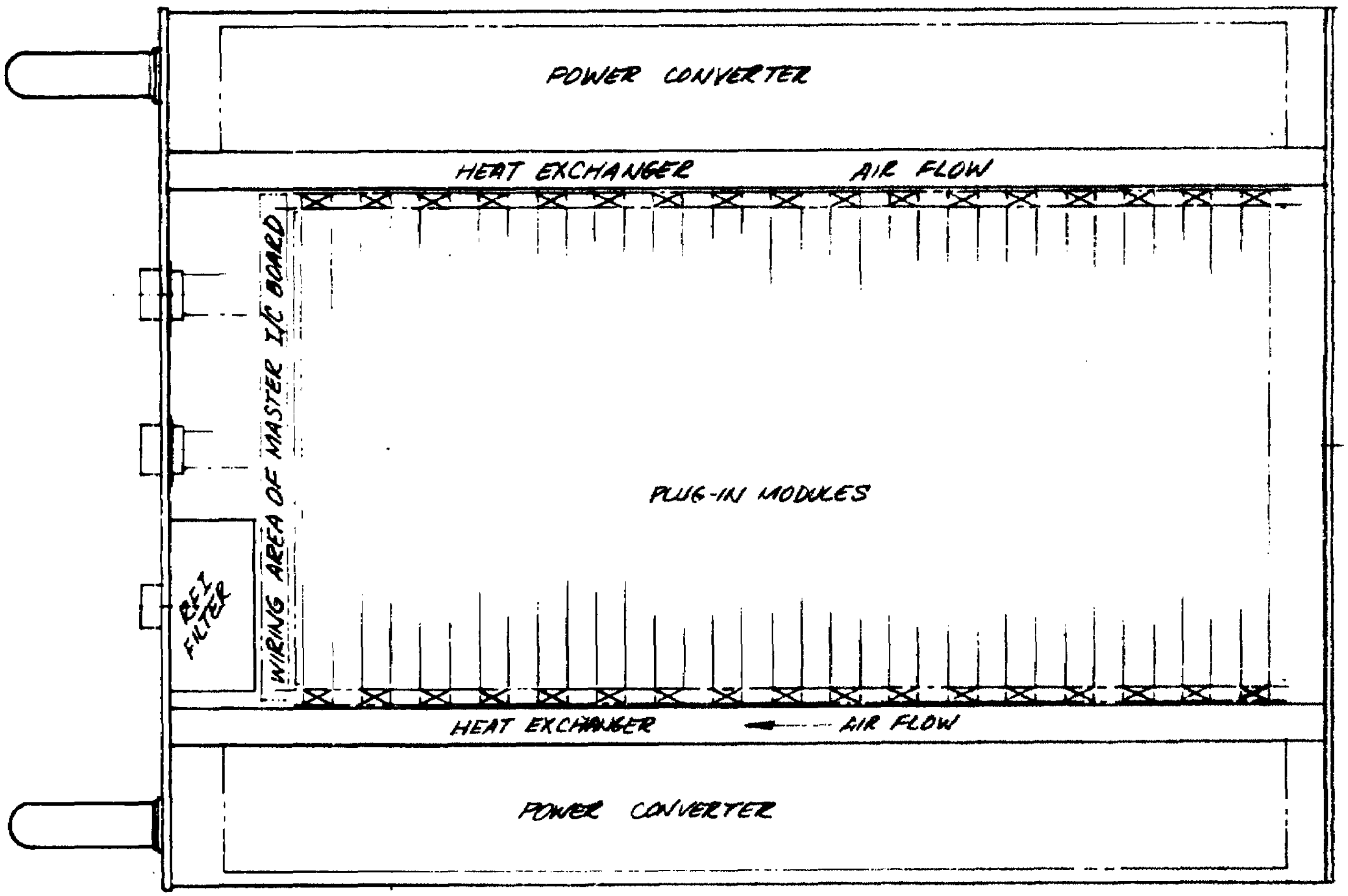


Figure 6-14. General Arrangement Avionics Processor and Controller

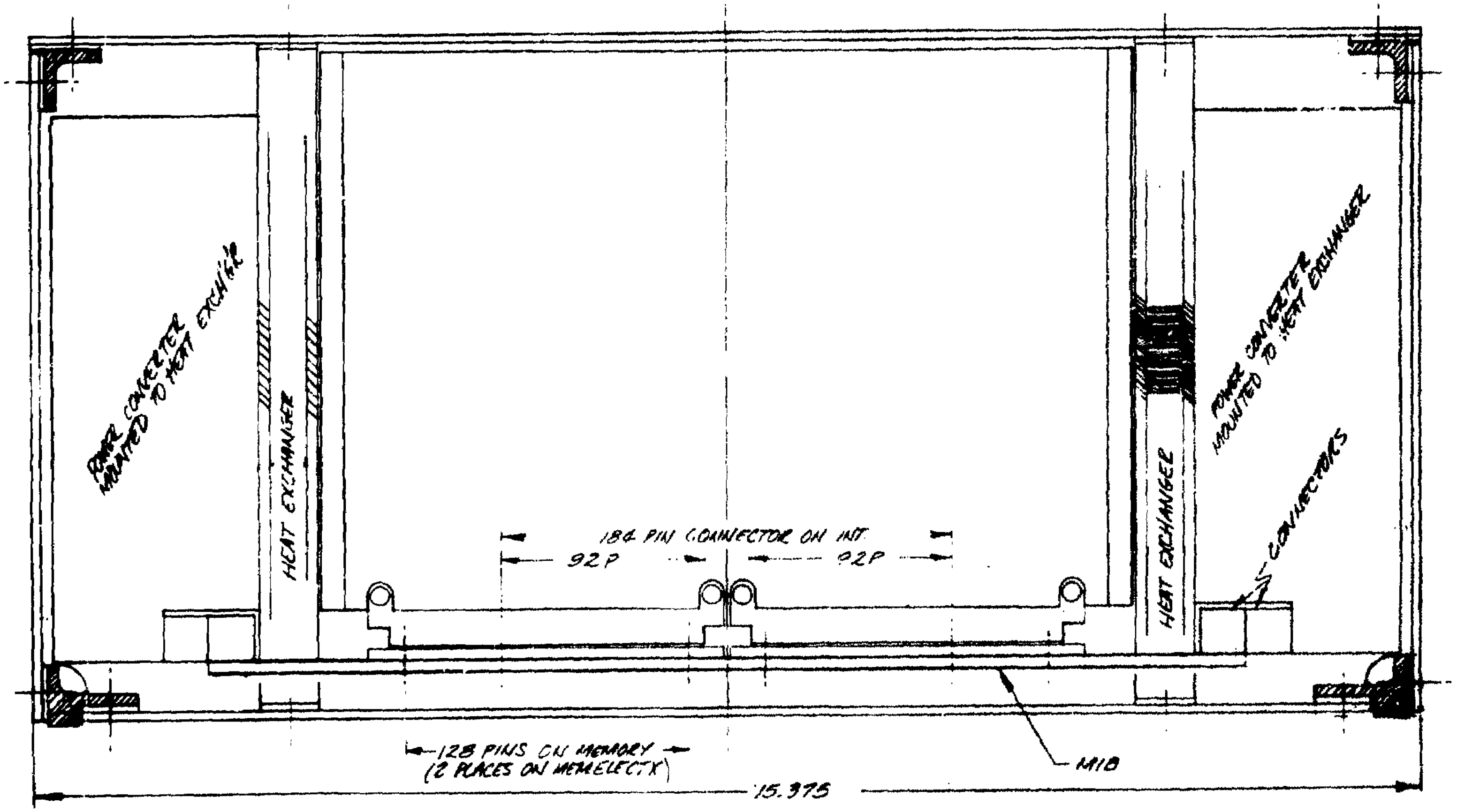

POWER CONVERTER MOUNTED TO HEAT EXCH'GR
HEAT EXCHANGER
184 PIN CONNECTOR ON INT
92P
92P
HEAT EXCHANGER
POWER CONVERTER MOUNTED TO HEAT EXCHANGER
CONNECTORS
128 PINS ON MEMORY (2 PLACES ON MEMELECTX)
MIB
15.375

Figure 6-15. Cross-Sectional View Avionics Processor and Controller

Multiprocessor 2:

4 Interpreters	100 watts
SWI	20 watts
2 Device Modules	20 watts
4 Memory Modules*	80 watts
Total	220 watts
with 2/3 efficiency in Power Supply	330 watts

6.4 SUMMARY OF PHYSICAL CHARACTERISTICS

The central processor was defined to consist of two multiprocessors, each in identical packaging units. The characteristics of each multiprocessor are summarized below:

Multiprocessor 1:

Size	1.41 cu ft
Weight	79 lb
Power	265 watts

Multiprocessor 2:

Size	1.41 cu ft
Weight	79 lb
Power	330 watts

*Note: The power dissipation of a memory module is approximately 20 watts when operating and 5 watts in standby. This power is approximately the same for the 4 K and 8 K memory modules. In MP1, since there is one more memory module than interpreters, one module will on the average be in standby.

7. SUMMARY AND CONCLUSIONS

The computational requirements for an advanced Avionics System were defined. In particular, the ASB avionics system was selected as representative of an advanced avionics system. The Burroughs multiprocessor concept was analyzed considering the requirements imposed on this design by the avionics system. It was found that in general the Burroughs multiprocessor offers a very flexible and adapuable design, however, it does have some limitations which were considered to be capable of being corrected.

The multiprocessor offers a wide possibility in the type of language choosen to run on the machine. It is suitable for emulation applications, execution of a higher level language, and optimization of a particular language to the application. The emulation of an IBM 4 π CP avionics computer was investigated in detail. It was estimated that a single interpreter's throughput capability when operating in such a mode would be approximately 79,000 operations/second for a typical avionics mix of operation (instruction) types. In addition, the optimization of this emulation through the use of macro instructions was investigated. It was found that for the same application, the effective throughput of an interpreter could be increased to 120,000 operations/ second. The throughput capability is considered somewhat low when compared to state-of-the-art avionics processors and this is the primary limitation of the Burroughs multiprocessor.

Various methods to increase the throughput capability of an interpreter were considered. Some of the more important ones are noted below:

1. Provide more A registers
2. Faster multiply algorithm
3. Modify logic to provide typically required functions such as: set carry latch for a carry overflow, more shift functions such as shift and spread sign, provide more conditions that can be set and more flexibility in testing multiple conditions.

The SWI module was analyzed to determine its timing, interface with memories/ devices, modularity, and failure tolerance. It was found that modularity and failure tolerance were not achievable with the present design. An alternate design was arrived at that essentially partitions the SWI into channels where each interpreter is dedicated to a channel. This new design enhances modularity and allows failure tolerance to be readily achieved through power control to individual channels. In addition the new design uses less modules and reduces the types of modules required.

Using the results of the avionics system requirements definition and the definition of the capabilities and limitations of the Burroughs multiprocessor modules, a configuration for the avionics application was arrived at. The processing requirements were analyzed to determine which computations should be performed in a central processor and which in a local processor at the subsystem level. As a result of this allocation analysis, three subsystems were allocated local processors: a penetration aids processor for the entire pen aids subsystem, a navigation processor for IMU control processing for the navigation subsystem, and a weapon delivery processor for the SRAM processing in the weapon delivery subsystem. The remainder of the processing functions were performed in the central processor. The resultant configuration for the central processor required two multiprocessors, one with three interpreters and the other with four interpreters.

The interface to the ASB avionics multiplex system was investigated and a multiplex interface controller module that functions as a device in the multiprocessor was defined. It was found that the inherent flexibility of the interpreter allowed it to readily perform I/O processing functions.

The failure detection and reconfiguration methods for the central processor were investigated and defined. Several modifications and additions were required in order to provide a failure tolerant system:

1. Real time clock that provides periodic interrupts added
2. Test counter added to interpreter
3. Interrupt module added
4. GC logic modified and moved to SWI
5. SWI partitioned per new design
6. Power switch added to SWI channels
7. Software detection, isolation, and reconfiguration programs added

It was determined that with the above modifications/additions, a failure tolerant multiprocessor could be achieved.

The executive required to operate the multiprocessor in a real time control avionics system was investigated and defined. It was determined that the Burroughs executive concept could be used with many simplifications to its basic structure.

A preliminary design of the central processor was performed based on state-of-the-art bipolar MSI logic and 2 mil plated wire memory technology. It was determined that the central processor would consist of two units, each housing one multiprocessor. Each multiprocessor occupies 1.41 cu ft and weighs 79 lbs. One multiprocessor dissipates 265 watts and the other 330 watts.

The conclusion reached from this study is that it is feasible to mechanize the ASB avionics computational system with the Burroughs multiprocessor concept and achieve a mechanization that is reasonable in its physical characteristics. This mechanization requires certain modifications as were noted above. In addition, improvements were noted that could further enhance the performance of the design and potentially improve the resultant physical characteristics.

REFERENCES

1. Davis, R. L., C. M. Campbell, S. Zucker; Aerospace Multiprocessor Interim Report, Burroughs Corporation, Feb 1972

2. Wehr, K. C., Technical Summary of the Interpreter-Based System, Burroughs Corporation, Jan 1971

3. Davis, R. L., and S. Zucker; Structure of a Multiprocessor Using Microprogrammable Building Blocks, NAECON '71 Record, pp 186-200.

4. Davis, R. L., S. Zucker, C. M. Campbell, A Building Block Approach to Multiprocessing, 1972 Spring Joint Computer Conference, pp 685-703.

5. Reigel, E. W., U. Faber, D. A. Fisher, The Interpreter - A Microprogrammable Building Block System, 1972 SJCC, pp 705-723.

6. L. F. Solberg, R. C. Ham, G. L. Kreglow, "B1 Common Language Study," North American Rockwell, Los Angeles Division, NA 71-605, October 1971.

7. Nielsen, W. C., Vere, S. A., Lauro, J. A., "Aerospace HOL Computer," AFAL-TR-72-292, Logicon, Inc., October 1972.

8. Mauck, E. A., and Dent, B. A., "Burroughs B6500/B7500 Stack Mechanism" Proc SJCC 1968, pp 245-251.

9. Keeler, F. S., etal "Computer Architecture Study" AF/SAMSO, Report TR 240, October, 1970.

10. C. R. Frost, Military CPU's, Datamation, July 15, 1970, pp 87-98.

11. Conti, C. J. "Concepts for Buffer Storage," Computer Group News, March 1969, pp 9-13.

12. D. H. Gibson "Considerations in Block Oriented Design," AFIPS, Spring 1967, pp 75-80.

13. P. M. Melliar - Smith "A Design for a Fast Computer for Scientific Calculations," AFIPS, Fall 1969, pp 201-208.

14. "Computer Aided Requirements Determination," Report No. C8-2778/301, North American Rockwell - Autonetics Division, November 1968.

15. "The Application of Information Transfer Techniques for Solving the Internal Communication Requirements of an Advanced Manned Bomber," Technical Report AFAL-TR-72-209, Vol 1, Radiation, Inc., September 1972.

16. "B-1 Multiplex Interface Module Preliminary Technical Requirements," B-1 Division, North American Rockwell, September 12, 1972.

17. Koczela, L. J., A Three Failure Tolerant Computer System, IEEE Transactions on Computers, November 1971.

18. Zucker, S., Aerospace Multiprocessor Executive, Burroughs Corp, Paoli, Pa., Technical Report AFAL-TR-72-144, April 1972.

DISTRIBUTION LIST

Contract F33615-72-C-1973

Address	No. of Copies
WPAFB ACTIVITIES	
AFAL/TSR WPAFB OH 45433	1
AFAL/AAM (Mr. J. Camp) WPAFB OH 45433	18
AFIT (Library) WPAFB OH 45433	2
ASD/YHEV (Mr. Jim Hutson) WPAFB OH 45433	2
2750ABW/SSL WPAFB OH 45433	1
OTHER ACTIVITIES	
HQ USAF/SAMID Wash DC 20330	1
AU Library Maxwell AFB AL 36112	1
Director Naval Research Lab Wash DC 20390	1
Commanding Officer Naval Avionics Facility 21st and Arlington Ave Indianapolis IN 46218	1
US Army Electronics R&D Lab Attn: Dr. H. Jacobs Ft Monmouth NJ 07703	1
Director, NSA R-13 Ft George Meade MD 20755	1

DDC 2
Cameron Station
Alexandria VA 22314

INDUSTRY

Control Data Corp 1
4130 Linden Ave
Dayton OH 45432

Hughes Aircraft Co 1
Aerospace Group
Culver City CA 90230

Honeywell 1
Military Products Group
2314 Standly Ave
Dayton OH 45404

IBM Corp 1
33 West First St
Dayton OH 45402

RCA 1
Aerospace Systems Division
Box 588
Burlington MA 01801

McDonnell Douglas Corp 1
333 West First St
Dayton OH 45402

Raytheon 1
333 West First St
Dayton, Ohio 45402

Westinghouse Electric Corp 1
Aerospace Division
Friendship International Airport
Box 746
Baltimore MD 21203

Litton Systems, Inc. 1
Guidance & COntrol System Division
5500
Canoga Ave
Woodland Hills CA 91364

Texas Instruments, Inc. 1
Equipment Group
Suite 205
3300 South Dixie Drive
Dayton OH 45439

General Electric Co 1
Aerospace & Defense Sales & Service
3430
South Dixie
Dayton OH 45439

Univac 1
Defence Systems Division
333 West First St
Dayton OH 45402

Burroughs Corp 1
Federal & Special Systems Group
Attn: D.F. Sullivan
Paoli PA 19301

Boeing Computing Systems 1
Attn: J.F. Cramer
8R-39 Mail Stop
Box 3707
Seattle WA 98124

Singer-Kearfott Division 1
Attn: M.G. Page
33 West First St
Dayton OH 45402

The Garrett Corp 1
333 West First St
Dayton OH 45402

Grumman Aircraft 1
333 West First St
Dayton OH 45402

Northrop Corp 1
379 West First St
Dayton OH 45402

Department of Transportation 1
Transportation Systems Center
Attn: Mr. G. Y. Wang
Cambridge, Mass.

National Aeronautics and Space Administration
Langley Research Center
Attn: Mr. L. Spencer
Hampton, Virginia 23365

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Autonetics Division of Rockwell International
3370 E. Miraloma Ave., Anaheim, CA 92803

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
Avionics Processor Controller Study, Volume 1, Technical Report

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report July 1972 - June 1973

5. AUTHOR(S) (First name, middle initial, last name)
L. J. Koczela

6. REPORT DATE
June 30, 1973

7a. TOTAL NO. OF PAGES
238

7b. NO. OF REFS
18

8a. CONTRACT OR GRANT NO.
F33615-72-C-1973

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
C72-812/201, Vol 1

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
AFAL-TR-73-203, Vol. 1

10. DISTRIBUTION STATEMENT
Distribution limited to U.S. Government Agencies only; test and evaluation results reported; February 1972. Other requests for this document must be referred to Air Force Avionics Laboratory (AAM), Wright-Patterson Air Force Base, OH45433.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
AFAL/AAM
WPAFB, OH 45433

13. ABSTRACT

This report presents the results of a study to configure an advanced multiprocessor for an avionics system. An advanced strategic bomber avionics system was selected as representative of an advanced avionics system application and computational requirements for this system were defined. The prototype laboratory version of an advanced multiprocessor developed by Burroughs Corporation under Air Force Avionics Laboratory sponsorship was examined and applied to the avionics system. It was found that the Burroughs multiprocessor offers a very flexible and adaptable design. Several improvements were noted to improve its performance and several design modifications were noted which are required in order to apply the design to the avionics system. The resultant configuration showed that mechanization of the computer system, using state-of-the-art technology, for an advanced strategic bomber avionics system is feasible with the Burroughs multiprocessor concept. This report is also being published as Autonetics internal report C72-812/201.

UNCLASSIFIED

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Avionics Systems						
Multiprocessing						
Computer Architecture						
Computer Organization						
Microprogramming						
Computational Requirements						
Failure Tolerant Systems						

Security Classification